

Proyecto Final de Grado



**IMPLEMENTACIÓN DE
UN CHATBOT
EDUCATIVO EN AWS
PARA MEJORA DE LA
COMUNICACIÓN EN
CENTROS EDUCATIVOS**

"Descubre la comodidad de obtener respuestas rápidas y soluciones eficientes!"

**Memoria PFG
Administración de Sistemas Informáticos en Red**

**Autor: Serhiy Holovasin
Tutora: María Jesús Sánchez García**

IES Infanta Elena (Galapagar-Madrid)

Curso 2023-2024

TÍTULO: Implementación de un ChatBot Educativo en AWS para mejora de la Comunicación en Centros Educativos

AUTOR: SERHIY HOLOVASIN SUSLOV

TUTOR DEL PROYECTO: MARÍA JESÚS SÁNCHEZ GARCÍA

FECHA DE LECTURA:

CALIFICACIÓN:

Fdo: SERHIY HOLOVASIN SUSLOV

Fdo: MARÍA JESÚS SÁNCHEZ GARCÍA

RESUMEN

La transformación digital y la preferencia por interacciones rápidas y eficientes han modificado la comunicación en el sector educativo, destacando la urgencia de adaptarse a las expectativas de las nuevas generaciones. Este proyecto introduce la creación e implementación de un chatbot educativo desplegado en la nube de Amazon Web Services (AWS), dirigido a optimizar la comunicación entre estudiantes y centros educativos, tales como institutos, colegios y universidades. Utilizando AWS para alojar y gestionar el chatbot, este proyecto aprovecha la escalabilidad, seguridad y alta disponibilidad que la plataforma ofrece, garantizando un servicio accesible las 24 horas del día. Desarrollado con Python y soportado por una interfaz web creada con HTML, CSS y JavaScript, el chatbot está diseñado para ofrecer respuestas inmediatas a consultas frecuentes, aliviar la carga administrativa y facilitar un acceso sin precedentes a la información educativa. A través de este enfoque, el proyecto busca satisfacer la demanda de interacciones digitales eficaces, marcando un hito en la evolución de la comunicación educativa y estableciendo un modelo de referencia para la integración tecnológica en centros educativos.

ABSTRACT

Digital transformation and the preference for quick and efficient interactions have changed communication within the educational sector, highlighting the urgency to adapt to new generations' expectations. This project introduces the creation and implementation of an educational chatbot deployed on Amazon Web Services (AWS) cloud, aimed at optimizing communication between students and educational centers, including institutes, colleges, and universities. By utilizing AWS to host and manage the chatbot, this project leverages the scalability, security, and high availability that the platform offers, ensuring a 24/7 accessible service. Developed with Python and supported by a web interface created with HTML, CSS, and JavaScript, the chatbot is designed to provide immediate responses to frequent inquiries, relieve administrative burden, and facilitate unprecedented access to educational information. Through this approach, the project seeks to meet the demand for effective digital interactions, marking a milestone in the evolution of educational communication and establishing a benchmark model for technological integration in educational centers.

AGRADECIMIENTOS

"Agradecer no cuesta nada pero vale mucho."

Este proyecto no hubiera sido posible sin la ayuda, apoyo y guía de muchas personas cuya colaboración ha sido invaluable.

Un agradecimiento especial a mi profesora y tutora, María Jesús Sánchez García, cuya dirección, paciencia y conocimiento fueron muy importantes para mi aprendizaje y progreso en este proyecto. Su apoyo y consejos me han permitido superar los obstáculos que enfrenté y han sido una fuente de inspiración constante.

También quiero agradecer especialmente a mis compañeros, amigos y todos los profesores del departamento por su ayuda y enseñanzas durante este curso.

Por supuesto, un enorme agradecimiento a mi familia. Su amor y comprensión han sido el apoyo, que me ha permitido llegar hasta aquí. Gracias por estar siempre ahí, en cada paso del camino, acompañándome en cada nuevo reto.

Finalmente, un sincero agradecimiento a la comunidad educativa por proporcionar las herramientas y recursos necesarios para mi formación.

¡A todos ustedes, gracias por hacer este viaje posible y tan especial!

COPYRIGHT



Esta obra se distribuye bajo una licencia Creative Commons.

Se permite la copia, distribución, uso y comunicación de la obra si se respetan las siguientes condiciones:

- Se debe reconocer explícitamente la autoría de la obra incluyendo esta nota y su enlace.
- La copia será literal y completa
- No se podrá hacer uso de los derechos permitidos con fines comerciales, salvo permiso expreso de los autores.

El texto precedente no es la licencia completa sino una nota orientativa de la licencia original completa(jurídicamente válida) que puede encontrarse en:

<https://creativecommons.org/licenses/by-nc-nd/4.0/deed.es>

ÍNDICE DE CONTENIDOS

| | |
|---|-----------|
| RESUMEN | 3 |
| ABSTRACT..... | 3 |
| AGRADECIMIENTOS..... | 4 |
| COPYRIGHT..... | 5 |
| ÍNDICE DE CONTENIDOS..... | 6 |
| ÍNDICE DE ESQUEMAS | 8 |
| GLOSARIO..... | 9 |
| 1. INTRODUCCIÓN | 14 |
| 1.1. OBJETIVOS | 14 |
| 1.2. MOTIVACIÓN | 16 |
| 1.3. ANTECEDENTES | 16 |
| 2. ESTIMACIÓN DE RECURSOS Y PLANIFICACIÓN..... | 18 |
| 3. DESARROLLO DE LA PRACTICA..... | 19 |
| 3.1. TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS | 19 |
| 3.2. ¿QUÉ CLOUD CONTRATAMOS Y PORQUE?..... | 21 |
| 3.3. DESARROLLO DE INTERFAZ WEB..... | 23 |
| 3.4. DESARROLLO DE CHATBOT | 26 |
| 3.5. CONFIGURACIÓN DE LA NUBE Y LAS ESPECIFICACIONES DE LAS ESTANCIAS | 27 |
| 3.6. ¿CÓMO VIAJAN LAS PREGUNTAS Y RESPUESTAS A TRAVÉS DE NUESTRA INFRAESTRUCTURA DE AWS?.... | 30 |
| 3.7. ESQUEMA BBDD - DIAGRAMA E/R | 31 |
| 4. PLAN DE PRUEBAS..... | 34 |
| 4.1. OBJETIVOS DE PLAN DE PRUEBAS | 34 |
| 4.2. TIPOS DE PRUEBAS | 34 |
| 4.3. CONCLUSIONES DE LA FASE DE PRUEBAS | 43 |
| 5. RESULTADO FINAL | 45 |

| | |
|---|-----------|
| 6. REFERENCIAS | 47 |
| 7. ANEXOS | 49 |
| 7.1. ANEXO: Estructura del Proyecto. Código. | 49 |
| 7.2. ANEXO: Código del Chatbot: Parte del Cliente | 51 |
| 7.3. ANEXO: Configuración del Backend en python_apps | 54 |
| 7.4. ANEXO : Modelos y Rutas de la API en python_apps | 56 |
| 7.5. ANEXO: Configuración de la Base de Datos y Modelos en python_apps | 59 |
| 7.6. ANEXO: Lógica del Chatbot..... | 62 |
| 7.7. ANEXO: Configuración de la Base de Datos y Generación de Datos en db_setup | 67 |
| 7.8. ANEXO: Adquisición de dominio | 75 |
| 7.9. ANEXO: Configuración de Frontend en AWS con S3 y CloudFront | 77 |
| 7.10. ANEXO: Construcción de Imagen Docker para el Chatbot..... | 83 |
| 7.11. ANEXO: Uso de la API de GPT-3.5 Turbo de OpenAI | 87 |
| 7.12. ANEXO: Configuración de Amazon ECS con Fargate y Application Load Balancer | 90 |
| 7.13. ANEXO: Esquema Gantt actualizado | 94 |

ÍNDICE DE ESQUEMAS

| | |
|--|----|
| Esquema 1: Primer diagrama de Gantt-SmartSheet..... | 18 |
| Esquema 2: Variedad de servicios dentro de AWS | 22 |
| Esquema 3: Pantalla de selección del rol..... | 23 |
| Esquema 4: Interfaz WEB..... | 24 |
| Esquema 5: Interfaz WEB con el ChatBot en uso. | 25 |
| Esquema 6: Interconexión de los servicios AWS | 27 |
| Esquema 7: Estructura de la Base de Datos..... | 31 |
| Esquema 8: Configuración y mapa de recursos de Application Load Balancer | 38 |
| Esquema 9: Configuración de las políticas de autoescalado..... | 39 |
| Esquema 10: Métricas de CPU en utilización en el momento. | 39 |
| Esquema 11: Métricas observadas en CloudWatch. | 40 |
| Esquema 12: Mapa de recursos de Application Load Balancer con autoescalado.. | 40 |
| Esquema 13: Métricas de CPU en utilización en el momento. | 41 |
| Esquema 14: Implementación de WebSockets..... | 42 |
| Esquema 15: Diagrama de Gantt-SmartSheet actualizada..... | 95 |

GLOSARIO

AWS (Amazon Web Services)

Una plataforma de servicios en la nube que ofrece soluciones de computación escalables y potentes. AWS proporciona una amplia gama de recursos como servidores, almacenamiento y bases de datos, disponibles bajo demanda y gestionados a través de Internet, permitiendo a las empresas y desarrolladores reducir costos y escalar aplicaciones de manera eficiente.

Escalabilidad

Capacidad de un sistema, red o proceso para manejar un creciente volumen de trabajo de manera eficiente, o su capacidad para ampliarse para acomodar ese crecimiento. En el contexto de la informática, se refiere a la habilidad de un sistema para incrementar su capacidad de carga y procesamiento agregando recursos, ya sea de forma horizontal (más máquinas) o vertical (más potencia en una sola máquina).

Alta Disponibilidad

Característica de un sistema que asegura un nivel acordado de rendimiento operacional, usualmente uptime, para un periodo de tiempo más alto de lo normal. En el contexto de servicios informáticos, se refiere a sistemas o componentes que son continuamente operativos durante un período extensamente definido. La alta disponibilidad busca evitar cualquier punto de fallo único, utilizando redundancias y técnicas de fallo seguro.

Python

Lenguaje de programación de alto nivel, interpretado, y de propósito general. Python es conocido por su facilidad de aprendizaje y lectura, su sintaxis clara y su comunidad amplia que soporta numerosas bibliotecas y frameworks. Es ampliamente utilizado en desarrollo web, análisis de datos, inteligencia artificial, ciencia computacional, y automatización.

HTML (Lenguaje de Marcado de Hipertexto)

Lenguaje de marcado estándar para documentos diseñados para ser mostrados en un navegador web. HTML describe la estructura de una página web semánticamente e incluye cues(pistas) para su apariencia visual. Se utiliza para crear y estructurar secciones, párrafos, encabezados, enlaces, y elementos de bloque para páginas web y aplicaciones.

CSS (Hojas de Estilo en Cascada)

Lenguaje de hojas de estilo utilizado para describir la presentación de un documento escrito en HTML o XML. CSS describe cómo los elementos deben ser presentados en pantalla, en papel, en el habla, o en otras formas de medios. Permite controlar la disposición de múltiples páginas web al mismo tiempo.

JavaScript

Lenguaje de programación interpretado que se puede utilizar para hacer que las páginas web sean interactivas (por ejemplo, tener controles deslizantes animados, mapas interactivos, y contenido en vivo). JavaScript es esencial en el desarrollo web como parte del trío de tecnologías estándar de la web, junto con HTML y CSS.

Docker

Docker es una plataforma de contenedores de código abierto que permite a los desarrolladores empaquetar aplicaciones y sus dependencias en un contenedor virtualizado, que puede ser ejecutado en cualquier sistema operativo que soporte [Docker](#). Esto simplifica la configuración, aumenta la portabilidad y asegura la consistencia del entorno de ejecución de las aplicaciones, independientemente de dónde se desplieguen. Docker es altamente popular en el desarrollo de software moderno por su eficiencia, facilidad de uso y la capacidad de aislar las aplicaciones, reduciendo los conflictos entre ellas y maximizando la utilización de los recursos del sistema.

IDE (Entorno de Desarrollo Integrado)

Un entorno de desarrollo integrado (IDE) es una aplicación de software que proporciona a los programadores y desarrolladores un conjunto de herramientas y funcionalidades integradas para escribir, modificar, probar y depurar su código de manera más eficiente. Un IDE típicamente incluye un editor de texto con resaltado de sintaxis y auto-completado, herramientas de construcción automática, un depurador, y a menudo integra sistemas de control de versiones. Algunos IDEs populares incluyen [Visual Studio Code](#), [Eclipse](#) y [IntelliJ IDEA](#). Estos entornos son especialmente útiles en el desarrollo de software ya que centralizan todas las herramientas necesarias en una sola interfaz, facilitando el desarrollo y mantenimiento de programas complejos.

CRUD

Es un acrónimo para Create, Read, Update, Delete, que representa las cuatro operaciones básicas utilizadas en el manejo de bases de datos y aplicaciones de software para gestionar datos. Estas operaciones permiten a los usuarios crear nuevos registros, leer o consultar registros existentes, actualizar registros anteriores y eliminar registros no deseados. Esta estructura es fundamental para el desarrollo de aplicaciones interactivas que requieren un manejo constante y sistemático de datos.

VPC (Virtual Private Cloud):

[VPC](#) es una sección aislada de la nube de AWS, en nuestro proyecto, es donde podemos lanzar recursos de AWS en una red virtual que nosotros mismos definimos. Ofrece control completo sobre el entorno de red virtual, incluyendo selección de rangos de IP, creación de

subredes, y configuración de tablas de rutas y redes privadas. En definitiva, es una versión virtual de una red física.

Amazon EC2 (Elastic Compute Cloud)

Amazon EC2 es un servicio de AWS que proporciona capacidad de computación escalable en la nube. Permite a los usuarios alquilar servidores virtuales (instancias) para ejecutar aplicaciones y administrar cargas de trabajo. EC2 ofrece un control completo sobre los sistemas informáticos en un entorno virtualizado, permitiendo elegir la configuración del hardware, el sistema operativo y el entorno de red, lo que facilita la escalabilidad y la gestión eficiente de la infraestructura de TI.

Máquina Virtual (VM)

Una máquina virtual es una representación digital o emulación de una computadora física. Funciona en un entorno aislado conocido como hipervisor, que permite ejecutar múltiples sistemas operativos simultáneamente en un solo host físico. Cada máquina virtual opera independientemente con su propio sistema operativo y recursos, tales como CPU, memoria y almacenamiento, asignados desde el hardware subyacente. Las máquinas virtuales son ampliamente utilizadas para la consolidación de servidores, pruebas de software, desarrollo de aplicaciones y despliegue de múltiples entornos operativos sin la necesidad de hardware adicional, ofreciendo una forma eficiente, escalable y segura de maximizar el uso de los recursos computacionales disponibles.

Amazon RDS (Relational Database Service)

Amazon RDS es un servicio gestionado que facilita la configuración, operación y escalado de bases de datos relacionales en la nube. Proporciona una capacidad rentable y redimensionable mientras maneja tareas de administración de bases de datos tediosas como el aprovisionamiento de hardware, la configuración de la base de datos, la realización de copias de seguridad y el parcheado. RDS soporta varios motores de bases de datos, incluyendo PostgreSQL, MySQL, MariaDB, Oracle y Microsoft SQL Server.

Amazon ELB (Elastic Load Balancing)

ELB es un servicio de AWS que distribuye automáticamente el tráfico de entrada de aplicaciones o sitios web entrantes a través de múltiples destinos, como instancias EC2, contenedores y direcciones IP, en varias zonas de disponibilidad. ELB aumenta la tolerancia a fallos de las aplicaciones al asegurar que solo las instancias sanas reciban tráfico y escalando la capacidad de manejo de tráfico según las demandas, lo que mejora la robustez y el rendimiento de la aplicación.

Amazon Route 53

Amazon Route 53 es un servicio de sistema de nombres de dominio (DNS) web escalable y altamente disponible ofrecido por AWS. Facilita a los desarrolladores y empresas una forma confiable y rentable de enrutar a los usuarios finales a aplicaciones de Internet al traducir nombres como www.ejemplo.com en direcciones IP numéricas como 192.0.2.1 que las computadoras usan para conectarse entre sí. Route 53 es bien conocido por su capacidad para manejar eficazmente tanto las necesidades de enrutamiento como las de registro de dominios.

Auto Scaling Group:

Un grupo de auto escalado permiten el escalado automático y la gestión de un grupo de instancias de Amazon EC2 que tratan el aumento o disminución del tráfico. Estos grupos ajustan automáticamente el número de instancias EC2 en funcionamiento según las políticas definidas, los horarios o la salud de las instancias.

IAM (Identity and Access Management)

IAM es un servicio de AWS que ayuda a controlar el acceso seguro a los recursos de AWS de manera eficaz. Permite gestionar usuarios, grupos, y permisos, y utilizar políticas para conceder o denegar acceso a los recursos de AWS a individuos o servicios. IAM es crucial para administrar la seguridad de las operaciones en la nube.

EBS (Elastic Block Store)

EBS proporciona volúmenes de almacenamiento en bloque de alto rendimiento para su uso con instancias EC2 en AWS. Los volúmenes EBS están diseñados para ser altamente disponibles y fiables, y pueden ser utilizados como almacenamiento principal de una instancia EC2 o para aplicaciones que requieren un acceso frecuente a disco y un rendimiento consistente.

Amazon S3 (Simple Storage Service)

Es un servicio de almacenamiento de objetos que proporciona escalabilidad, disponibilidad de datos, seguridad y rendimiento. Este servicio permite a los usuarios almacenar y recuperar cualquier cantidad de datos desde cualquier ubicación en la web. Es ampliamente utilizado para hacer copias de seguridad y recuperación de desastres, alojar aplicaciones web, y como un repositorio de datos para análisis. Amazon S3 se destaca por su modelo de pago por uso, su alta durabilidad y su capacidad para gestionar datos a gran escala.

Amazon CloudFront

Es un servicio de red de entrega de contenido (CDN) que distribuye contenido con alta transferencia y baja latencia a usuarios en todo el mundo. CloudFront se integra con otros servicios de Amazon como Amazon S3, EC2, y Route 53 para ofrecer un sistema eficiente y

seguro de entrega de contenido estático y dinámico, incluyendo sitios web, aplicaciones y transmisiones en vivo. CloudFront también proporciona protecciones de seguridad integradas, como la negociación SSL para el tráfico encriptado y la integración con AWS Shield para protección contra DDoS.

CDN (Content Delivery Network)

Es una red de servidores distribuidos estratégicamente en diferentes ubicaciones geográficas diseñada para optimizar la entrega de contenido en internet, como páginas web, imágenes, videos y otros archivos. Al almacenar copias del contenido en varios servidores, un CDN permite a los usuarios acceder a este desde el servidor más cercano, reduciendo la latencia y mejorando la velocidad de carga. Además, los CDN aumentan la seguridad y la capacidad de manejo de tráfico, ayudando a mitigar ataques de denegación de servicio (DDoS) y otros tipos de amenazas.

1. INTRODUCCIÓN

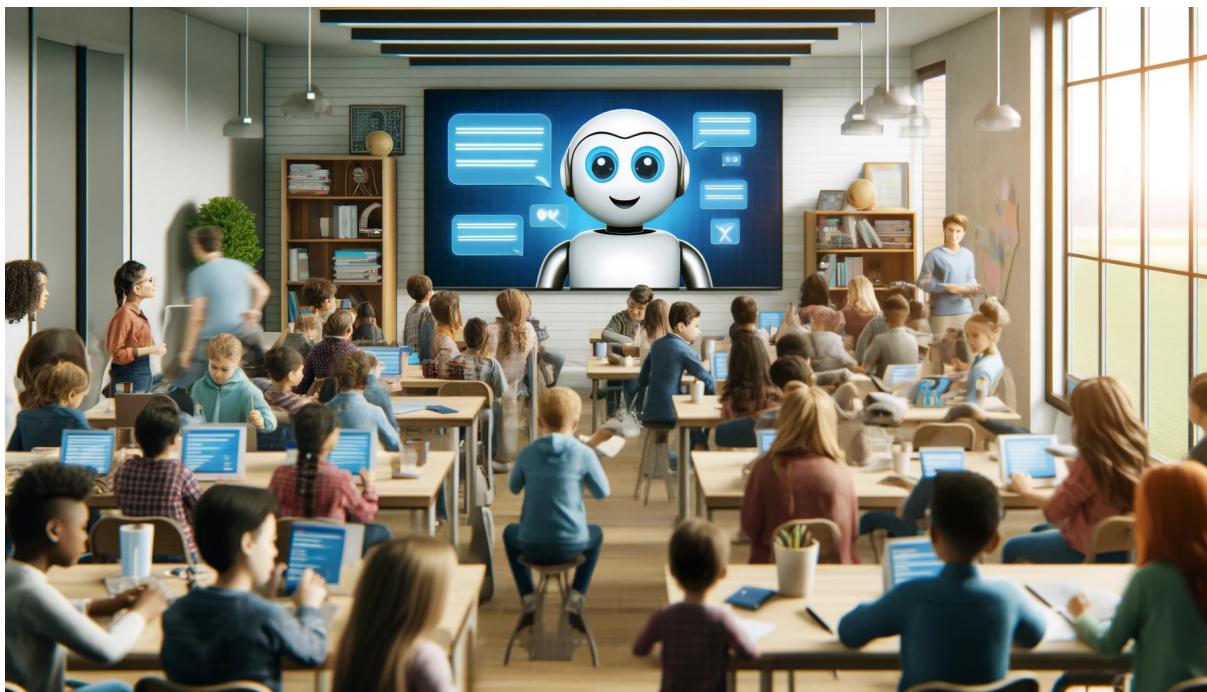


Imagen 1. “La clase del futuro”

La era digital ha reconfigurado las expectativas de interacción dentro de numerosos sectores, siendo el educativo uno de los más impactados. Estudiantes y centros educativos buscan ahora medios de comunicación que ofrezcan inmediatez, flexibilidad y personalización. Ante este panorama, surge la necesidad de adaptar y mejorar los canales de comunicación existentes para satisfacer las demandas de las nuevas generaciones. Este proyecto se centra en el desarrollo e implementación de un chatbot educativo alojado en Amazon Web Services, diseñado para revolucionar la manera en que los centros educativos interactúan con su estudiantes, profesores, trabajadores administrativos y familias (padres y/o tutores) de los estudiantes.

1.1. OBJETIVOS

El objetivo principal del proyecto es implementar un chatbot educativo en AWS que mejore la comunicación entre los usuarios y los centros educativos, tales como institutos, colegios y universidades. Los objetivos específicos incluyen:

1. *Proporcionar una plataforma de comunicación instantánea y accesible las 24 horas del día, adaptándose a la inmediatez requerida por los estudiantes.*

La generación actual de estudiantes espera respuestas rápidas y accesibilidad constante a la información. Un chatbot que ofrece interacciones instantáneas y está disponible todo el día responde a estas expectativas, facilitando un medio de comunicación que es tanto conveniente como eficiente. Esto no solo mejora la satisfacción del usuario sino que también potencia el compromiso con el centro educativo.

2. *Utilizar la infraestructura de AWS para garantizar la escalabilidad, seguridad y alta disponibilidad del chatbot, permitiendo un servicio ininterrumpido y confiable.*

La selección de AWS como plataforma principal para el chatbot es estratégica. Su infraestructura robusta asegura que el chatbot pueda escalar según las variaciones en la demanda de uso, manteniendo un alto nivel de rendimiento sin importar el número de usuarios simultáneos. Además, AWS ofrece avanzadas características de seguridad para proteger tanto la información del usuario como las interacciones con el chatbot, garantizando la confidencialidad y la integridad de los datos. La alta disponibilidad, por su parte, asegura que el servicio esté operativo en todo momento, minimizando cualquier tiempo de inactividad y ofreciendo una experiencia de usuario continua.

3. *Reducir la carga administrativa en el personal educativo mediante la automatización de respuestas a consultas frecuentes.*

Uno de los problemas persistentes en los centros educativos es la alta carga de trabajo administrativo relacionada con la gestión de consultas estudiantiles. Automatizando las respuestas a las preguntas más frecuentes, el chatbot alivia significativamente esta carga, permitiendo que el personal dedique más tiempo a tareas críticas que requieren atención humana. Esto no solo optimiza los recursos sino que también mejora la calidad del servicio educativo al centrar el esfuerzo humano en áreas donde es más valioso.

4. *Ofrecer una solución adaptable y escalable que pueda ser personalizada según las necesidades específicas de cada centro educativo.*

Cada centro educativo tiene su conjunto único de necesidades, políticas y desafíos. Un chatbot personalizable y escalable permite una implementación flexible que se puede adaptar a estas condiciones específicas. Esto asegura que la solución no solo sea relevante y efectiva en un amplio rango de contextos educativos sino que también pueda evolucionar con el tiempo para abordar nuevos requerimientos o integrar mejoras. La capacidad de adaptación y escalabilidad es, por lo tanto, fundamental

para el éxito a largo plazo del proyecto y su capacidad para servir eficazmente a la comunidad educativa en su conjunto.

El objetivo final de este proyecto es conseguir:

“Que hablar con una escuela o universidad sea tan fácil y rápido como chatear con un amigo. Vamos a proyectar un chatbot que vive en la nube de AWS, listo para ayudarle con cualquier pregunta, día y noche, sin hacerle esperar. Queremos que este chatbot haga más que solo responder preguntas; queremos que haga la vida en la escuela más fácil para todos, desde estudiantes hasta profesores. Esto significa menos papeleo y más tiempo para lo que realmente importa. Al final, este chatbot no es solo una herramienta inteligente; es el primer paso para hacer que la educación sea más amigable, personal y completamente al alcance de tu mano.”

1.2. MOTIVACIÓN

La motivación de este proyecto nace de la necesidad de modernizar la comunicación en la educación, alineándola con las expectativas digitales actuales. Se elige AWS por su capacidad de soportar un sistema como el chatbot, que requiere alta disponibilidad, seguridad y flexibilidad para adaptarse a diferentes cargas de trabajo. Este enfoque busca ofrecer respuestas rápidas y acceso constante a la información, clave para estudiantes que viven en un mundo digital. Además, al automatizar las respuestas a preguntas comunes, se reduce la presión sobre el personal educativo, liberando su tiempo para tareas más críticas. El proyecto apunta a hacer la comunicación educativa más eficiente, segura y adaptable, utilizando tecnología avanzada para satisfacer las necesidades de una nueva era educativa.

1.3. ANTECEDENTES

Los chatbots, como interfaces conversacionales impulsadas por inteligencia artificial, han transformado diversas industrias, incluida la educación, al facilitar interacciones dinámicas y personalizadas que simulan conversaciones humanas.



La historia de los chatbots comenzó en 1966 con ELIZA, creado por Joseph Weizenbaum, que es considerado el primer chatbot capaz de simular una conversación. Desde entonces, los chatbots han evolucionado considerablemente, incorporando tecnologías avanzadas de procesamiento de lenguaje natural para mejorar su capacidad de entender y

responder de manera más humana (Planeta Chatbot, 2019).

En el ámbito educativo, los chatbots han mostrado un potencial significativo para revolucionar las prácticas de enseñanza y aprendizaje. Según un estudio detallado en el documento (“Chatbots en Educación”, 2023), estos sistemas no solo facilitan la gestión administrativa al responder preguntas frecuentes, sino que también actúan como asistentes pedagógicos, proporcionando tutorías personalizadas y apoyo continuo a los estudiantes.

La integración de chatbots en la educación responde a la necesidad de herramientas que soporten un aprendizaje más interactivo y personalizado. Artículos como el publicado en (UTEC, 2021 y aulasimple, 2021) destacan cómo los chatbots pueden mejorar significativamente la accesibilidad y la eficiencia de la educación, permitiendo a los estudiantes explorar nuevos contenidos de manera autónoma y recibir respuestas instantáneas a sus consultas.

Este proyecto se fundamenta en investigaciones anteriores que han explorado el uso de chatbots en mejorar la eficiencia de las interacciones educativas y simplificar el acceso a la información. Estos estudios revelan que los chatbots son técnicamente viables y ofrecen una interfaz directa y eficaz para responder consultas estándar, lo que los hace ideales para entornos educativos donde se requiere gestionar grandes volúmenes de preguntas frecuentes de manera rápida y precisa. Tal implementación no solo reduce la carga administrativa sobre el personal docente sino que también asegura que los estudiantes reciban respuestas inmediatas a sus inquietudes básicas, facilitando un ambiente de aprendizaje más fluido y accesible.



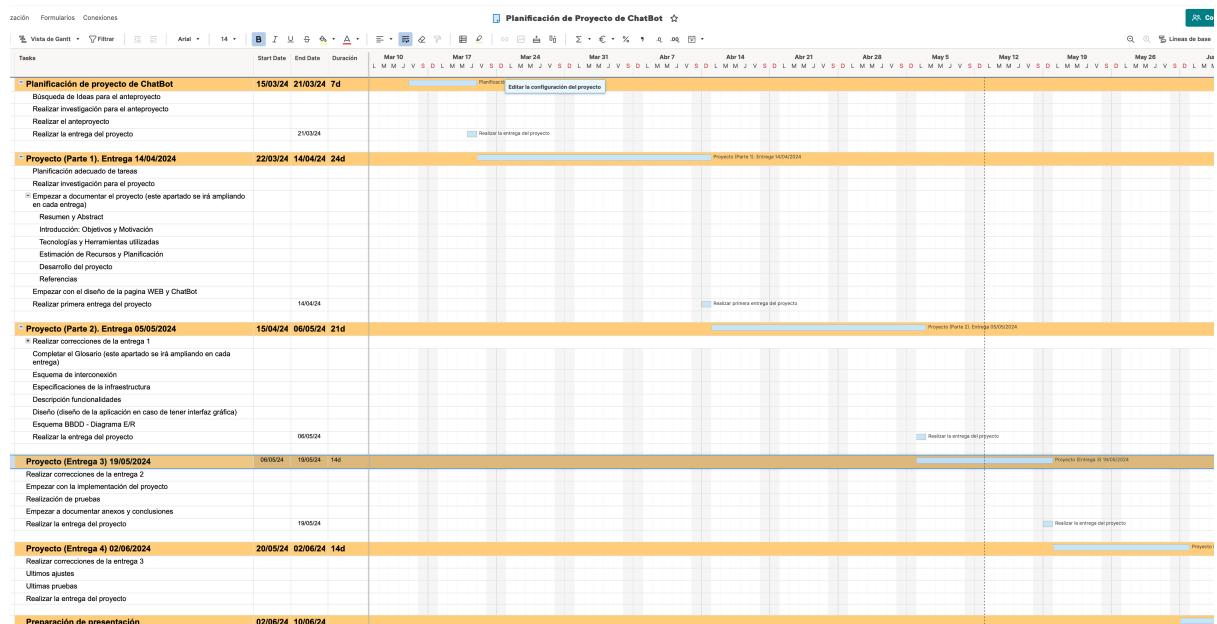
Imagen 2. "Evolución de la IA y los Chatbots"

2. ESTIMACIÓN DE RECURSOS Y PLANIFICACIÓN

Para desarrollar y desplegar el chatbot educativo en AWS de manera efectiva, se requieren recursos específicos tanto en hardware y software como en servicios en la nube. En términos de hardware y software, es imprescindible contar con una computadora que disponga de acceso a internet y la capacidad adecuada para el desarrollo de software. Además, es esencial una suscripción activa a AWS que permita el acceso a servicios clave como EC2, RDS, Elastic Load Balancing y Route 53. Respecto al software de desarrollo, se utilizará un entorno de desarrollo integrado (IDE), como Visual Studio Code, que es ideal para programar en Python. También se empleará Docker, una herramienta crucial para la creación de contenedores que alojarán el chatbot, garantizando así un entorno de ejecución consistente.

Adicionalmente, se contará con recursos de aprendizaje y soporte como la documentación oficial de AWS, tutoriales en línea, foros y comunidades, que serán de gran ayuda para resolver dudas específicas que surjan durante el desarrollo.

La planificación del proyecto se llevará a cabo mediante un enfoque de gestión de proyectos ágil utilizando Smartsheet, una herramienta poderosa para la planificación y seguimiento de proyectos. Smartsheet permitirá crear un cronograma detallado, asignar tareas, establecer plazos y colaborar en tiempo real.



Esquema 1: Primer diagrama de Gantt-SmartSheet

Al inicio del proyecto, hemos establecido un diagrama de Gantt-SmartSheet que muestra la planificación inicial de todas las tareas necesarias para el desarrollo del chatbot educativo.

Este esquema nos ayuda a visualizar y seguir el progreso de cada etapa del proyecto desde el principio. Al final del proyecto, un esquema de Gantt actualizado compara la planificación inicial con cómo se desarrollaron realmente las actividades. Este enfoque nos permitirá ver las diferencias entre la planificación y la ejecución, y entender mejor los ajustes que se hicieron durante el proyecto.

La fase de diseño y especificación abarcará la definición de los requerimientos funcionales y técnicos del chatbot, así como el diseño de la arquitectura general del sistema y la selección de tecnologías. Seguidamente, en la fase de desarrollo, se configurará el entorno de desarrollo y pruebas, se desarrollará el backend y la lógica del chatbot, y se integrará con la base de datos y otros servicios de AWS.

La fase de pruebas incluirá pruebas unitarias y de integración para asegurar la correcta funcionalidad del sistema, seguido de ajustes basados en los resultados obtenidos. El despliegue y configuración se realizará en el entorno de producción de AWS, donde también se configurará y gestionará la escalabilidad y la carga. Finalmente, en la fase de evaluación y ajustes, se evaluará el funcionamiento del chatbot y se optimizará el rendimiento, corrigiendo errores que se identifiquen. Todo el proceso será documentado meticulosamente en Smartsheet para garantizar que las configuraciones y mejoras puedan ser referenciadas en el futuro.

Este cronograma es una estimación inicial y puede requerir ajustes conforme avance el proyecto y se encuentren imprevistos. La organización y gestión eficaz del tiempo serán fundamentales, y las revisiones regulares serán esenciales para evaluar el progreso y hacer los ajustes necesarios en la planificación.

3. DESARROLLO DE LA PRACTICA

El desarrollo de la práctica se enmarca en un proceso detallado y estructurado que involucra múltiples fases, desde el diseño inicial hasta la implementación y despliegue final. Este proceso no solo se centra en el desarrollo de software, sino también en la integración eficiente de diversas tecnologías y plataformas para crear una solución robusta, escalable y segura.

3.1. TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS

Para el desarrollo del chatbot educativo, se ha priorizado un enfoque simple y eficaz tanto en términos de diseño como de funcionalidad, utilizando tecnologías estándar de la web y

servicios de backend avanzados, así como integración con la API de OpenAI para procesamiento de lenguaje natural.

Frontend:

- **HTML:** Se emplea para estructurar una interfaz de usuario clara y concisa del chatbot. La simplicidad del diseño HTML facilita la demostración y el uso intuitivo del chatbot por parte de los usuarios.
- **CSS:** Utilizado para aplicar estilos básicos que mejoran la legibilidad y la estética de la página, sin sobrecargarla con elementos visuales complejos. Esto asegura que la interfaz sea accesible y fácil de navegar durante las demostraciones.
- **JavaScript:** Implementado para añadir interactividad básica a la página, permitiendo la comunicación fluida entre el usuario y el chatbot. JavaScript maneja las entradas de los usuarios y muestra las respuestas del chatbot de manera eficiente, manteniendo la interfaz receptiva y ligera.



Backend:



• **Python:** Escogido por su sintaxis intuitiva y el amplio soporte para bibliotecas de NLP, facilitando la integración del chatbot con tecnologías avanzadas de IA.

- **FastAPI:** Este framework moderno es utilizado por su capacidad para manejar solicitudes asincrónicas rápidamente, lo cual es esencial para un rendimiento óptimo del chatbot en tiempo real.
- **Docker:** Empleado para encapsular la aplicación, asegurando que el chatbot pueda ser desplegado y ejecutado consistentemente en cualquier entorno sin problemas de configuración.

Conexión con OpenAI:

- La integración con la API de OpenAI permite que el chatbot acceda a capacidades avanzadas de procesamiento de lenguaje



natural, enriqueciendo las interacciones del chatbot con respuestas más coherentes y contextualmente apropiadas.

Servicios de AWS:

- Se utilizan para alojar la aplicación y gestionar la infraestructura necesaria de manera eficiente y segura, permitiendo que el chatbot escala según las necesidades del usuario y maneje variadas cargas de trabajo sin interrupciones.

Este diseño y elección de tecnologías garantizan que el chatbot sea no solo funcional y fácil de usar, sino también robusto y escalable, con una interfaz de usuario diseñada específicamente para facilitar demostraciones y pruebas de usuario.

3.2. ¿QUÉ CLOUD CONTRATAMOS Y PORQUE?

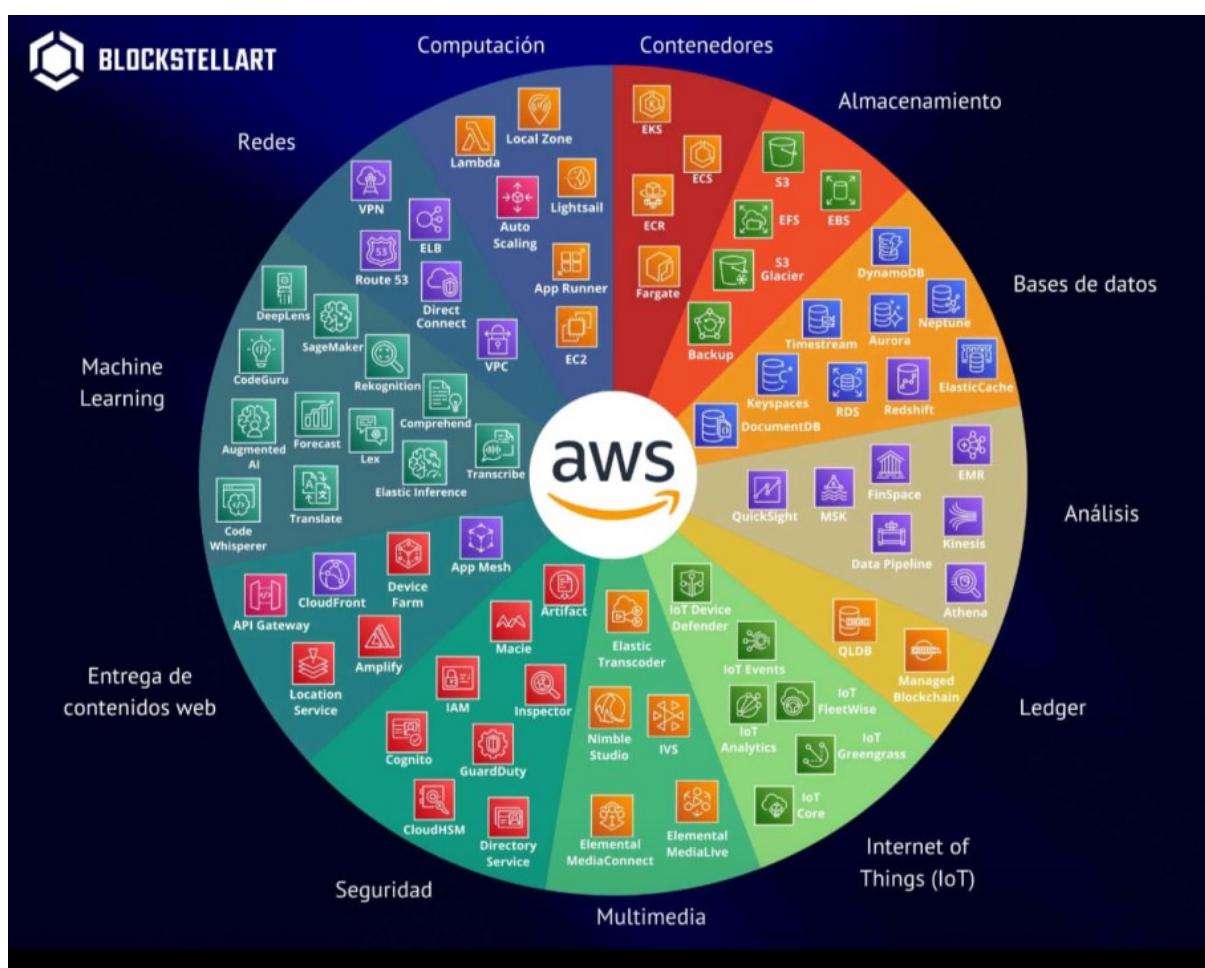
Para el alojamiento y la gestión de la infraestructura del chatbot educativo, se eligió Amazon Web Services como la plataforma de servicios en la nube. La elección de AWS se basó en varios factores críticos que se alinean con las necesidades del proyecto y garantizan su éxito.



Escalabilidad: AWS ofrece una amplia gama de recursos computacionales que se pueden escalar fácilmente. Esta característica es esencial para adaptar la capacidad de servicio del chatbot en función del crecimiento de la demanda o de picos inesperados de tráfico, asegurando que el sistema permanezca eficiente bajo cualquier carga de trabajo.

Fiabilidad y Alta Disponibilidad: AWS mantiene una red de centros de datos distribuidos globalmente, lo que permite implementar aplicaciones y datos en múltiples ubicaciones físicas. Esto no solo mejora la latencia general al estar más cerca de los usuarios finales, sino que también incrementa la redundancia y la continuidad del servicio en caso de fallo en uno de los centros de datos.

Variedad de Servicios Integrados: AWS proporciona una suite integrada de servicios que se pueden utilizar para desarrollar, desplegar y gestionar aplicaciones complejas. Esto incluye computación, almacenamiento, bases de datos, balanceo de carga, y más. La disponibilidad de estos servicios integrados simplifica el diseño de la arquitectura y reduce los tiempos de integración y depuración.



Esquema 2: Variedad de servicios dentro de AWS

Seguridad: AWS es reconocido por su robusto modelo de seguridad, que ofrece funcionalidades clave como la identificación y autenticación de acceso, cifrado de datos en tránsito y en reposo, y otras políticas de seguridad que ayudan a proteger y gestionar la

infraestructura. La seguridad es especialmente crítica para aplicaciones educativas donde la protección de datos de los usuarios es prioritaria.

Costo-Eficiencia: Utilizar AWS permite una reducción significativa en los costos de infraestructura al eliminar la necesidad de inversiones iniciales en hardware y reducir los costos operativos asociados con el mantenimiento de centros de datos físicos. Además, el modelo de pago por uso de AWS permite que solo se pague por los recursos que efectivamente se utilizan, lo que ayuda a mantener el proyecto dentro del presupuesto.

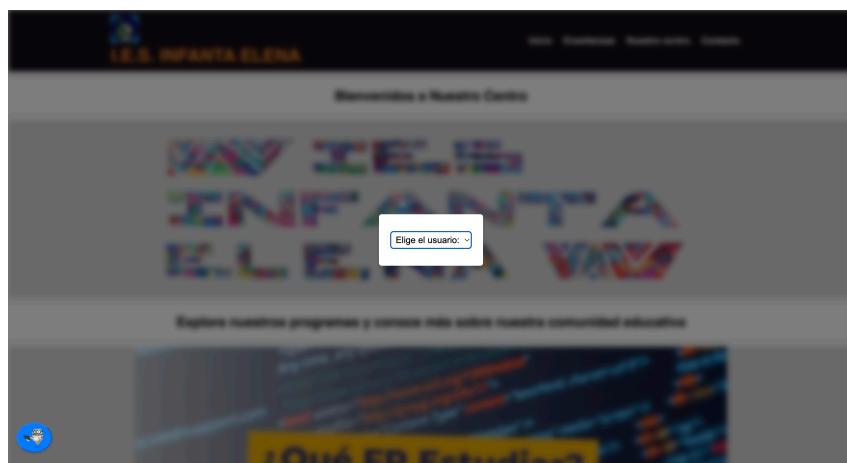
Estos factores hacen de AWS la plataforma ideal para desplegar el chatbot educativo, proporcionando una base sólida y flexible que apoya las necesidades del proyecto y garantiza su operatividad eficaz y segura.

3.3. DESARROLLO DE INTERFAZ WEB

La interfaz web del ChatBot Educativo ha sido diseñada para ofrecer una experiencia de usuario personalizada y relevante, adaptándose a las necesidades específicas de alumnos, profesores y padres. Este diseño permite que cada tipo de usuario acceda a información y funcionalidades ajustadas a sus roles particulares, mejorando así la comunicación y la eficiencia de las interacciones.

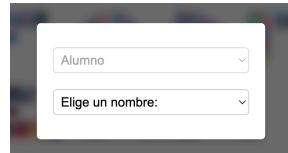
Descripción detallada:

Al ingresar a la plataforma, el usuario se enfrenta inicialmente con una pantalla de selección de rol, donde puede elegir entre '*Alumno*', '*Profesor*', o '*Padres*'. Esta elección determina la naturaleza de la interacción con el chatbot, asegurando que la información proporcionada sea pertinente y útil para el usuario.

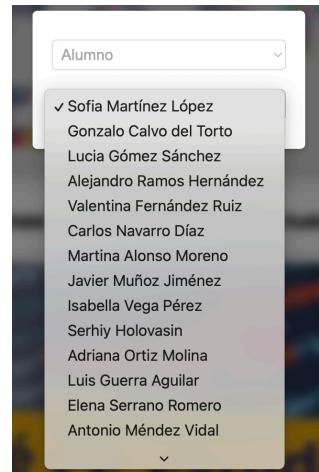


Esquema 3: Pantalla de selección del rol.

1. *Selección de Rol:* Cada usuario debe autenticarse seleccionando su rol específico, lo que influye directamente en el tipo de preguntas que el ChatBot puede resolver y los datos a los que puede acceder.



2. *Elección de Usuario:* Tras seleccionar el rol, se presenta al usuario una lista de nombres correspondiente a su categoría. Los padres deben seleccionar el nombre de su hijo o hija para proceder. Este paso es crucial para garantizar que la información mostrada sea relevante y específica para cada usuario.



3. *Acceso y Uso del Chat:* Una vez autenticados, los usuarios acceden a la página principal del ChatBot, donde pueden interactuar directamente con el sistema para obtener respuestas a sus consultas. La personalización del prompt del chatbot según el rol garantiza que las respuestas sean ajustadas y contextualizadas, mejorando la precisión de la información proporcionada.

La implementación de esta funcionalidad requiere una gestión de sesiones en el backend, asegurando que los datos de usuario sean almacenados y recuperados de manera eficiente durante las sesiones activas.

Bienvenidos a Nuestro Centro

I.E.S. INFANTA ELENA

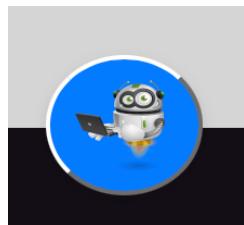
Explora nuestros programas y conoce más sobre nuestra comunidad educativa

© 2024 Centro Educativo. Todos los derechos reservados.

Esquema 4: Interfaz WEB

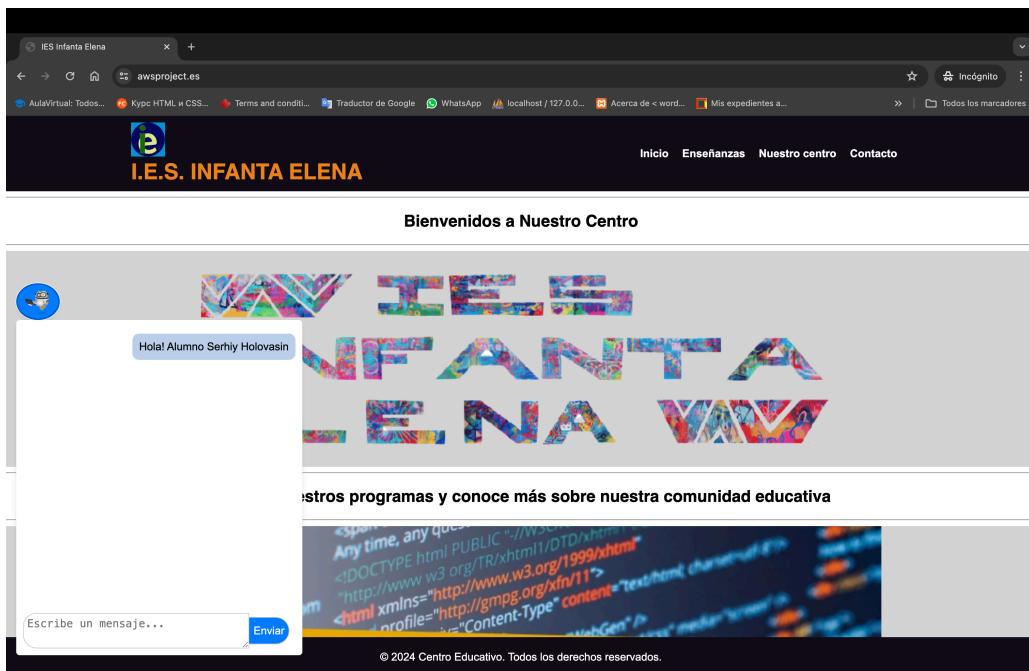
Se diseñó una interfaz de usuario utilizando HTML, CSS y JavaScript. Esta interfaz incluye :

- Un encabezado con el logo del centro educativo
- Navegación básica con enlaces a las páginas de “Inicio”, “Enseñanza”, “Nuestro centro” y “Contacto”
- Contenido de la pagina de inicio
- Un pie de página con derechos de autor.



La interfaz web incorpora un botón de chat que permite a los usuarios interactuar con el chatbot. Este botón activa una ventana de chat que se despliega para facilitar la comunicación directa con el chatbot, mejorando la accesibilidad y la experiencia del usuario.

El chatbot se integra en la página web mediante un contenedor de chat interactivo, donde los usuarios pueden escribir sus preguntas y recibir respuestas. La ventana de chat es completamente funcional y está diseñada para ser intuitiva y fácil de usar con visualización adaptada a PC, tablet y móvil.



Esquema 5: Interfaz WEB con el ChatBot en uso.

Enlace de la pagina: www.awsproject.es

Para la información mas detallada puede consultar:

► [ANEXO 7.2 : Codigo del chatbot: Parte del Cliente](#)

3.4. DESARROLLO DE CHATBOT

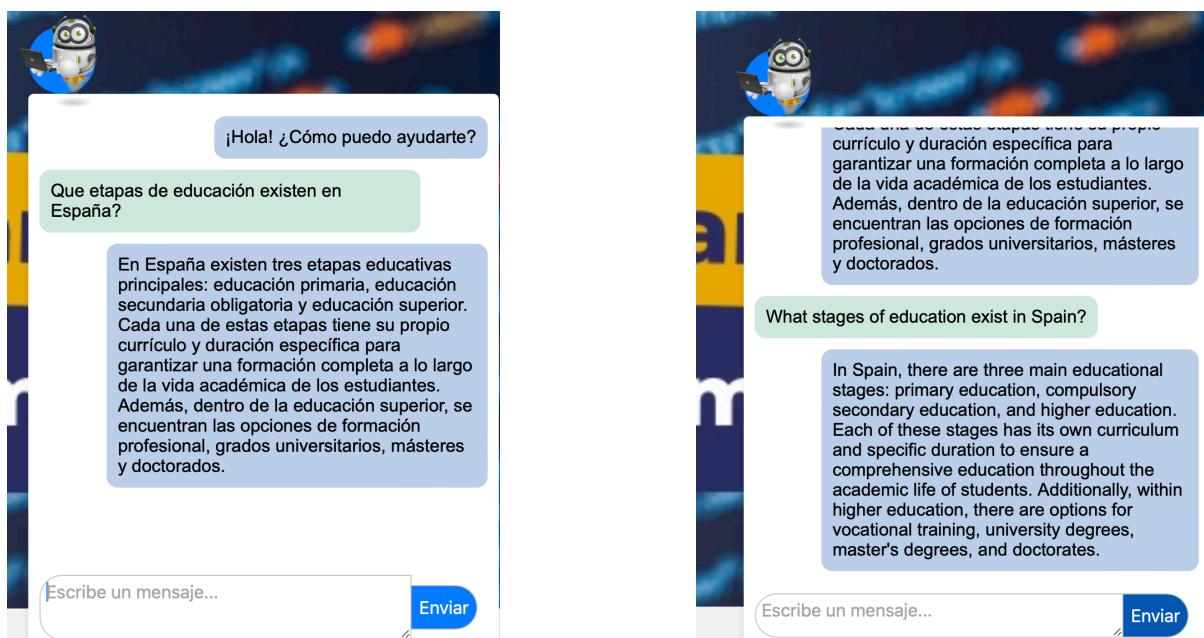
Se ha desarrollado un chatbot con python, utilizando tecnologías de procesamiento de lenguaje natural proporcionadas por la API de OpenAI. Este chatbot es capaz de responder preguntas y participar en conversaciones con los usuarios, proporcionando información relevante y asistencia en tiempo real.

Se estableció un servidor backend utilizando FastAPI, que gestiona las solicitudes de la interfaz de usuario. FastAPI fue elegido por su rendimiento y facilidad para crear APIs robustas y eficientes.

```
python_apps > app > main.py > ...
1  from fastapi import FastAPI, HTTPException
2  from fastapi.middleware.cors import CORSMiddleware
3  from pydantic import BaseModel
4  from dotenv import load_dotenv
5
6  import openai
7  import os
8
```

Se definieron rutas específicas en FastAPI para manejar las interacciones con el chatbot. Esto incluye una ruta POST que recibe las preguntas de los usuarios y devuelve respuestas generadas por el chatbot.

CORS Middleware: Se configuró un middleware de CORS en FastAPI para permitir solicitudes de origen cruzado, asegurando que la interfaz web pueda comunicarse sin problemas con el servidor backend.



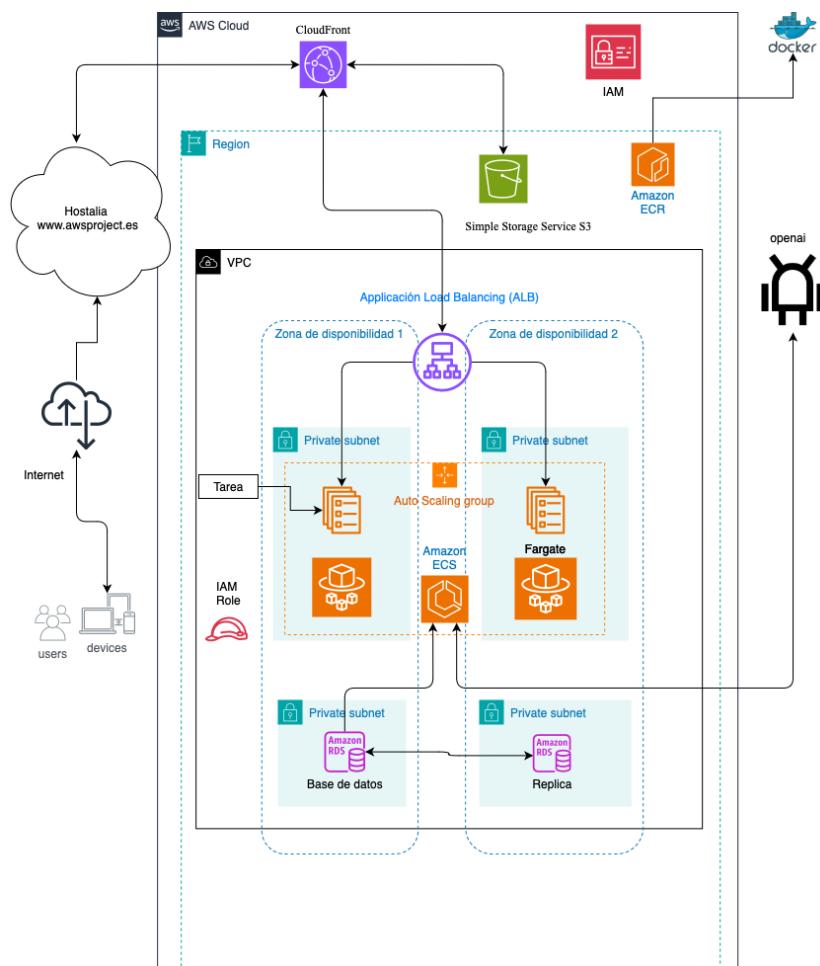
Se llevaron a cabo pruebas para asegurar que la interfaz web y el chatbot funcionen correctamente. Estas pruebas verificaron la correcta recepción de las solicitudes por parte de FastAPI y la adecuada respuesta del chatbot a las preguntas de los usuarios.

Hasta el momento, el proyecto ha logrado establecer una interfaz web interactiva y un chatbot funcional integrado mediante FastAPI. El sistema está diseñado para proporcionar asistencia en tiempo real a los usuarios, mejorando significativamente la interacción entre el centro educativo y su comunidad. La infraestructura actual establece una sólida base para futuras expansiones y mejoras del chatbot, incluyendo la capacidad de respuesta del sistema.

→ [ANEXO 7.3: Configuración del Backend en python_apps](#)

3.5. CONFIGURACIÓN DE LA NUBE Y LAS ESPECIFICACIONES DE LAS ESTANCIAS

La arquitectura del chatbot implementado en la nube se detalla a través de un diagrama que muestra cómo se integran los componentes de AWS para ofrecer un servicio robusto y seguro.



Esquema 6: Interconexión de los servicios AWS

En la fase inicial del proyecto, el esquema y las especificaciones de las instancias son provisionales y están diseñadas para ser ajustadas conforme avanza el desarrollo y se identifican nuevas necesidades o problemas técnicos.

Las instancias estarán configuradas para soportar los diversos componentes del chatbot, incluyendo servidores web y de aplicaciones. Se seleccionaran basándose en la necesidad de manejar múltiples solicitudes de manera simultánea y responder de manera eficiente, asegurando una experiencia de usuario fluida y sin interrupciones.

El frontend de la aplicación está alojado en Amazon S3, que proporciona un servicio de almacenamiento de objetos escalable y de alta disponibilidad. S3 sirve los archivos estáticos del frontend directamente a los usuarios finales, lo que garantiza tiempos de carga rápidos y una experiencia de usuario mejorada.

También se utiliza Amazon CloudFront, una red de entrega de contenido (CDN), para distribuir el contenido estático del frontend a los usuarios finales. CloudFront mejora la velocidad de carga de las páginas al cachear el contenido en múltiples ubicaciones geográficas, lo que reduce la latencia al servir el contenido desde la ubicación más cercana al usuario.

El dominio personalizado *awsproject.es* se gestiona a través de Hostalia, donde se han configurado los registros DNS necesarios para apuntar a la distribución de CloudFront. Además, se ha implementado un certificado SSL para asegurar la comunicación cifrada, mejorando la seguridad y la confianza del usuario en la aplicación.

Para garantizar la seguridad y la correcta redirección del tráfico, se han establecido políticas de redirección HTTP en CloudFront y S3, asegurando que todo el tráfico se maneje a través de conexiones seguras.

Esta configuración esta descrita en el:

- [ANEXO 7.8: Adquisición de dominio](#)
- [ANEXO 7.9: Configuración de Frontend en AWS con S3 y CloudFront](#)

En la etapa inicial, el backend de este proyecto se ejecuta en un entorno local, lo cual facilita el desarrollo y las pruebas rápidas de nuevas funcionalidades. En la etapa final, se despliega, utilizando los servicios de AWS junto con tecnologías de contenerización para maximizar la eficiencia y la escalabilidad dentro de los servicios y límites disponibles del entorno de laboratorio gratuito, garantizando al mismo tiempo que el sistema sea funcional y efectivo.

Los detalles específicos sobre la configuración y los pasos para su implementación final en AWS se detallan en:

- ➡ [ANEXO 7.12: Configuración de Amazon ECS con Fargate y Application Load Balancer](#)

Detalles de la implementación del backend, basado en los servicios de AWS:

1. *ECS y Fargate para la gestión de contenedores:* El backend estará desplegado utilizando Amazon Elastic Container Service con la opción Fargate, que permite ejecutar contenedores sin necesidad de administrar servidores o clústeres de máquinas virtuales. Fargate simplifica la configuración y el escalado, gestionando automáticamente la asignación de CPU y memoria, la programación de contenedores y la orquestación.
2. *Base de Datos en Docker utilizando SQLite:*
 - *SQLite en Docker:* Optamos por SQLite como nuestra solución de base de datos debido a su simplicidad y la capacidad de ejecutarse dentro de un contenedor Docker sin necesidad de configuraciones de servidor complejas. Esto es ideal para un entorno de laboratorio, ya que SQLite maneja todas las operaciones de base de datos con un overhead mínimo y no requiere gestión activa.
 - *Integración con Backend:* La base de datos SQLite está integrada con el backend en ECS, permitiendo que el ChatBot acceda y gestione datos de usuario y sesión de manera eficiente y segura.
3. *Balanceo de carga con Application Load Balancer:* Se implementará un Application Load Balancer para distribuir el tráfico entrante entre las instancias de ECS de manera eficiente. El ALB mejora la disponibilidad y la tolerancia a fallos del backend, proporcionando una distribución inteligente basada en la salud y el rendimiento de las instancias.
4. Los grupos de seguridad y las políticas de IAM serán fundamentales para definir y restringir el acceso a las instancias y otros recursos de AWS, asegurando que solo el tráfico autorizado y seguro pueda interactuar con el sistema.

3.6. ¿CÓMO VIAJAN LAS PREGUNTAS Y RESPUESTAS A TRAVÉS DE NUESTRA INFRAESTRUCTURA DE AWS?

Cuando un usuario entra en la página web del centro educativo, su solicitud es atendida por Amazon CloudFront, que sirve el contenido estático alojado en un Amazon S3 bucket. Este contenido incluye los archivos HTML, CSS y JavaScript necesarios para interpretar la página y habilitar la funcionalidad del chatbot. A medida que el usuario interactúa con el chatbot ingresando preguntas, el frontend captura estas interacciones mediante JavaScript.

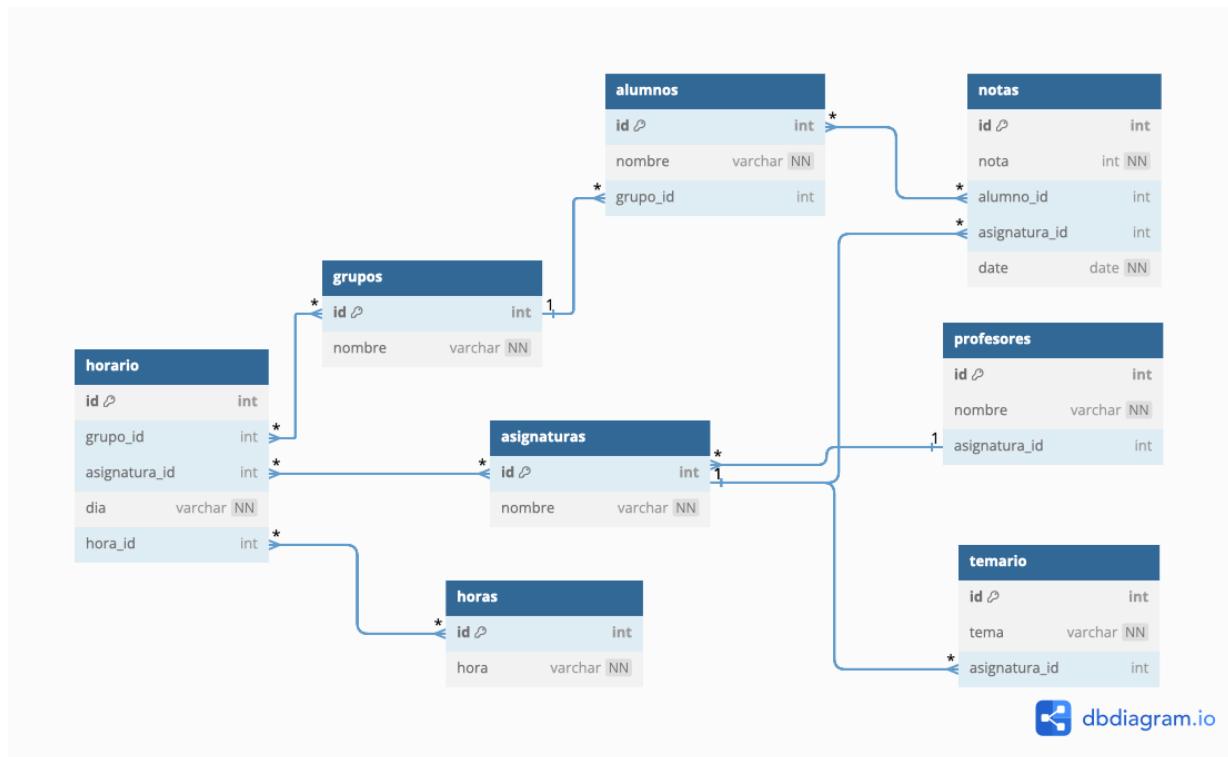
Los mensajes del usuario se envían al backend a través de solicitudes HTTP POST, que son dirigidas por Amazon CloudFront para mejorar la velocidad y la eficiencia de la entrega. Estas solicitudes son primero procesadas por un Elastic Load Balancer, que distribuye el tráfico entrante entre varias zonas de disponibilidad para balancear la carga y asegurar la alta disponibilidad. El backend se ejecuta en un entorno serverless proporcionado por AWS Fargate y ECS, donde se despliegan contenedores Docker que contienen la aplicación del chatbot. Este diseño serverless asegura que los recursos se escalen automáticamente en respuesta a las variaciones en el tráfico, optimizando así el uso de recursos y la eficiencia operativa.

Las solicitudes son procesadas por la aplicación backend, que utiliza la API de OpenAI para interpretar las preguntas del usuario y generar respuestas adecuadas. Esta integración permite al chatbot responder de manera inteligente y contextual, imitando una conversación humana fluida. Además, para preguntas que involucran datos específicos del centro educativo, como horarios de clases o detalles administrativos, el backend consulta una base de datos replicada en múltiples zonas de disponibilidad. Esta configuración de base de datos replicada no solo ofrece una recuperación rápida ante fallos sino también garantiza que la información esté siempre disponible y actualizada, independientemente de la carga o posibles problemas en una zona.

Finalmente, las respuestas generadas, ya sea a través de OpenAI o recuperadas de la base de datos, son enviadas de vuelta al frontend donde JavaScript actualiza dinámicamente la interfaz del chatbot para mostrar la respuesta al usuario. Este flujo no solo proporciona una experiencia de usuario rápida y confiable sino que también utiliza la infraestructura de AWS para garantizar seguridad, escalabilidad y continuidad del servicio.

3.7. ESQUEMA BBDD - DIAGRAMA E/R

El diseño de la base de datos para el proyecto del chatbot educativo está orientado a facilitar el acceso y la gestión de información clave sobre los horarios, asignaturas, alumnos, profesores, temarios y evaluaciones dentro de un centro educativo. La base de datos estructurada permite al chatbot interactuar de manera eficiente con los datos para responder a consultas específicas de los usuarios, ofreciendo una herramienta de apoyo significativa tanto para estudiantes como para el personal docente y administrativo.



Esquema 7: Estructura de la Base de Datos

La base de datos está compuesta por varias tablas que se relacionan entre sí:

- *Horario*: Enlaza grupos de estudiantes con asignaturas específicas y las horas a las que se imparten, permitiendo consultas sobre el calendario académico diario o semanal.
- *Grupos*: Almacena información sobre los diferentes grupos o clases de estudiantes, facilitando agrupaciones en consultas.
- *Asignaturas*: Incluye todas las asignaturas que se imparten, lo que permite al chatbot responder sobre detalles específicos de cada asignatura.

- *Alumnos*: Contiene información sobre los alumnos y a qué grupo pertenecen, importante para consultas personalizadas.
- *Profesores*: Detalla qué profesor enseña cada asignatura, permitiendo al chatbot informar sobre los docentes de cada materia.
- *Temario*: Registra los temas específicos de cada asignatura, útil para consultas sobre el contenido del curso.
- *Notas*: Mantiene registros de las calificaciones de los alumnos, proporcionando una base para consultas sobre rendimiento académico.

Basándose en esta estructura de la base de datos, el chatbot puede ser consultado con una variedad de preguntas, tales como:

“*¿Como me puedes ayudar?*”
 “*¿Saca mi nota media?*”
 “*¿Como puedo mejorar en Sistemas?*”
 “*¿Qué asignaturas tengo hoy?*”
 “*¿A qué hora es la clase de Sistemas el viernes?*”
 “*Muéstrame el horario del grupo Asir para tal fecha.*”
 “*Lista los alumnos del grupo ASIR.*”
 “*¿Quién enseña Redes ?*”
 “*¿Qué asignaturas imparte el profesor Fulanito?*”
 “*Dame información sobre la asignatura de Aplicaciones Web.*”
 “*¿Cuál fue mi nota en Bases de Datos tal fecha?*”
 “*¿Cómo voy en Hardware este trimestre?*”
 “*Lista mis calificaciones de este año.*”
 “*¿Qué temas cubre el curso de DAW?*”
 “*¿Qué voy a aprender en el curso de DAM este año?*”

El desarrollo y la integración del chatbot con esta base de datos requieren que se configuren adecuadamente las conexiones y las consultas SQL para recuperar información en respuesta a estas preguntas de manera eficiente y segura. El uso de técnicas de procesamiento de lenguaje natural y aprendizaje automático, como las proporcionadas por la plataforma de OpenAI, permitirán al chatbot interpretar las preguntas de los usuarios y convertirlas en consultas de base de datos pertinentes, facilitando así respuestas rápidas y precisas que mejoren la experiencia educativa de los usuarios.

Este ejemplo es una representación simplificada y fundamentalmente didáctica, diseñada para demostrar la funcionalidad básica del chatbot en interacciones específicas con los usuarios. Este esquema es expansible y puede ser enriquecido con muchas otras tablas y relaciones para adaptarse a necesidades más complejas y variadas de los centros educativos.

A continuación, se presentan algunas tablas adicionales que podrían incorporarse en futuras expansiones del sistema:

- *Eventos Escolares*: Esta tabla podría contener información sobre eventos próximos como excursiones, reuniones de padres, o eventos deportivos, permitiendo al chatbot informar a los estudiantes y padres sobre actividades importantes.
- *Recursos Educativos*: Una tabla destinada a almacenar recursos como libros, artículos, videos y enlaces a materiales educativos recomendados por profesores, facilitando que el chatbot sugiera recursos de aprendizaje pertinentes a las consultas de los alumnos.
- *Asistencia*: Registrar la asistencia diaria de los estudiantes a clases, lo que podría ayudar a los administradores y profesores a mantener un seguimiento más riguroso y automatizado de la presencia estudiantil.
- *Feedback de Cursos*: Almacenar evaluaciones y comentarios de los estudiantes sobre los cursos y profesores, proporcionando una retroalimentación valiosa que el chatbot podría recopilar de manera anónima y estructurada.

Esta modularidad y capacidad de expansión aseguran que el sistema pueda evolucionar junto con los requerimientos del centro educativo y las tecnologías emergentes.

También iré adaptando y refinando las preguntas y funcionalidades del chatbot durante las fases de desarrollo y pruebas, que nos permitirá ajustar la aplicación a las necesidades reales de los usuarios y optimizar la interacción con la base de datos.

4. PLAN DE PRUEBAS

Este capítulo se centrará en el conjunto de estrategias y métodos de pruebas empleados para garantizar que el chatbot educativo no solo cumpla con los requisitos funcionales, sino que también ofrezca una experiencia de usuario óptima y confiable. Se explorarán diferentes tipos de pruebas aplicadas en el proceso de desarrollo y cómo cada una contribuye a la creación de un sistema robusto y eficiente.

4.1. OBJETIVOS DE PLAN DE PRUEBAS

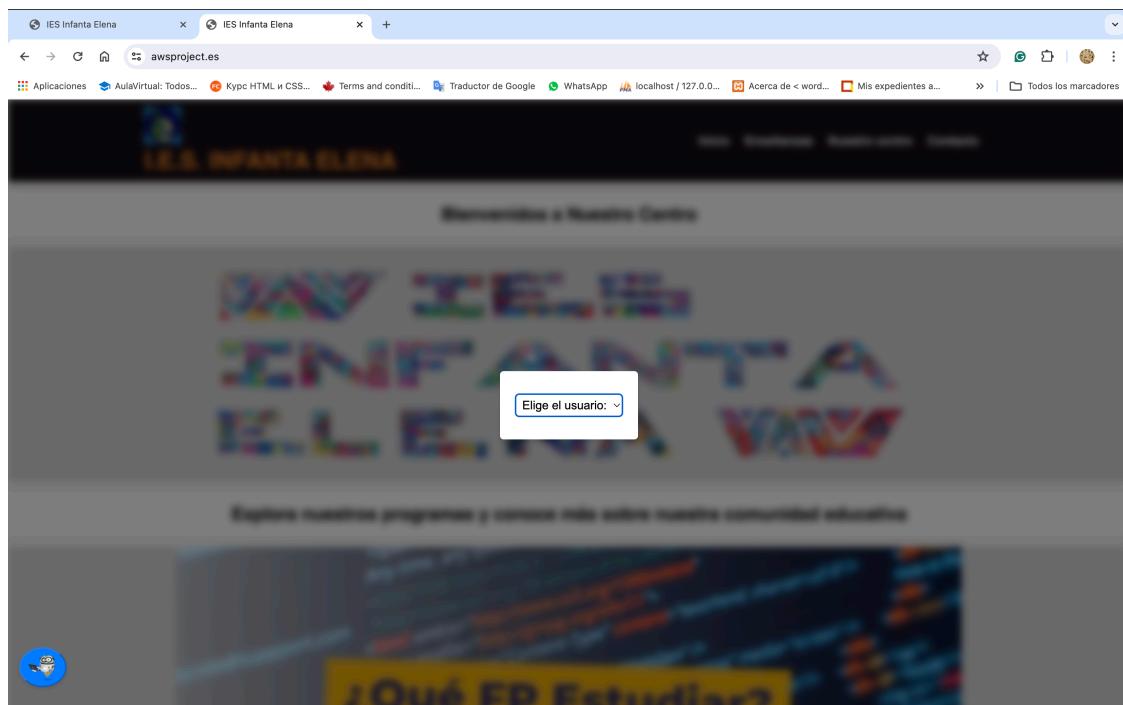
1. Validar la funcionalidad en diferentes plataformas y dispositivos.
2. Asegurar que el chatbot responde correctamente a las consultas de los usuarios.
3. Comprobar la usabilidad del sistema para garantizar una experiencia de usuario óptima.
4. Evaluar la capacidad del sistema para escalar y manejar un aumento en la carga de trabajo.

4.2. TIPOS DE PRUEBAS

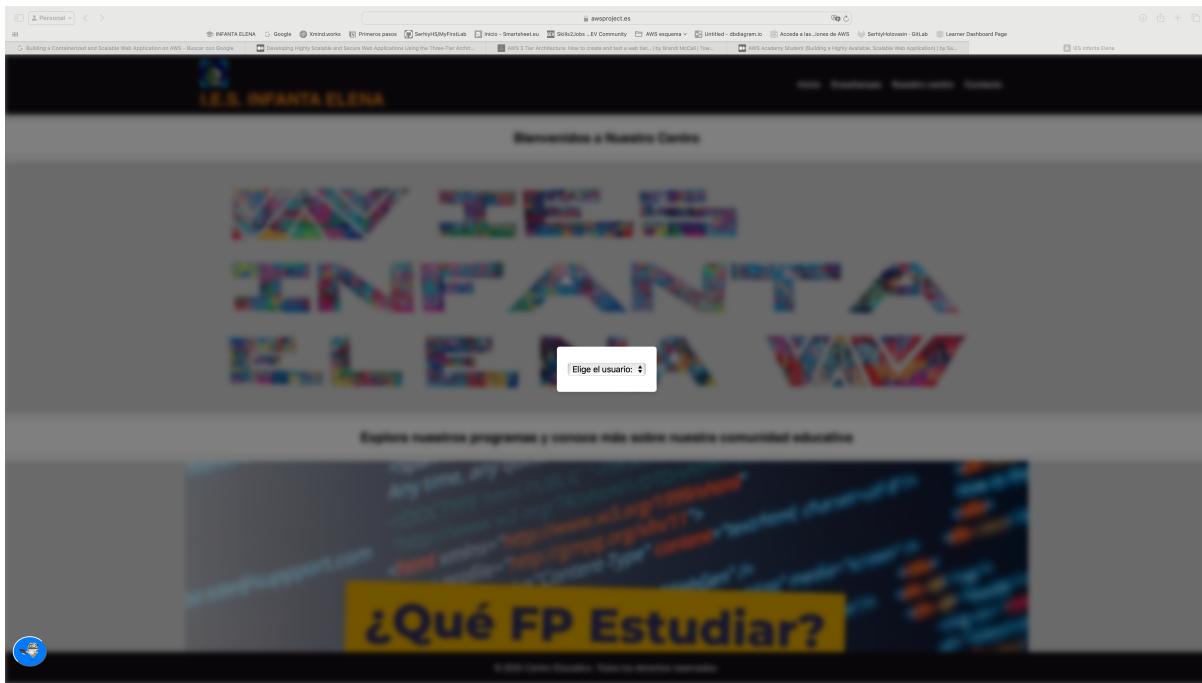
- *Pruebas de Compatibilidad:*

Compatibilidad entre Navegadores: Asegurar que el chatbot funcione correctamente en diferentes navegadores y dispositivos, proporcionando una experiencia de usuario consistente.

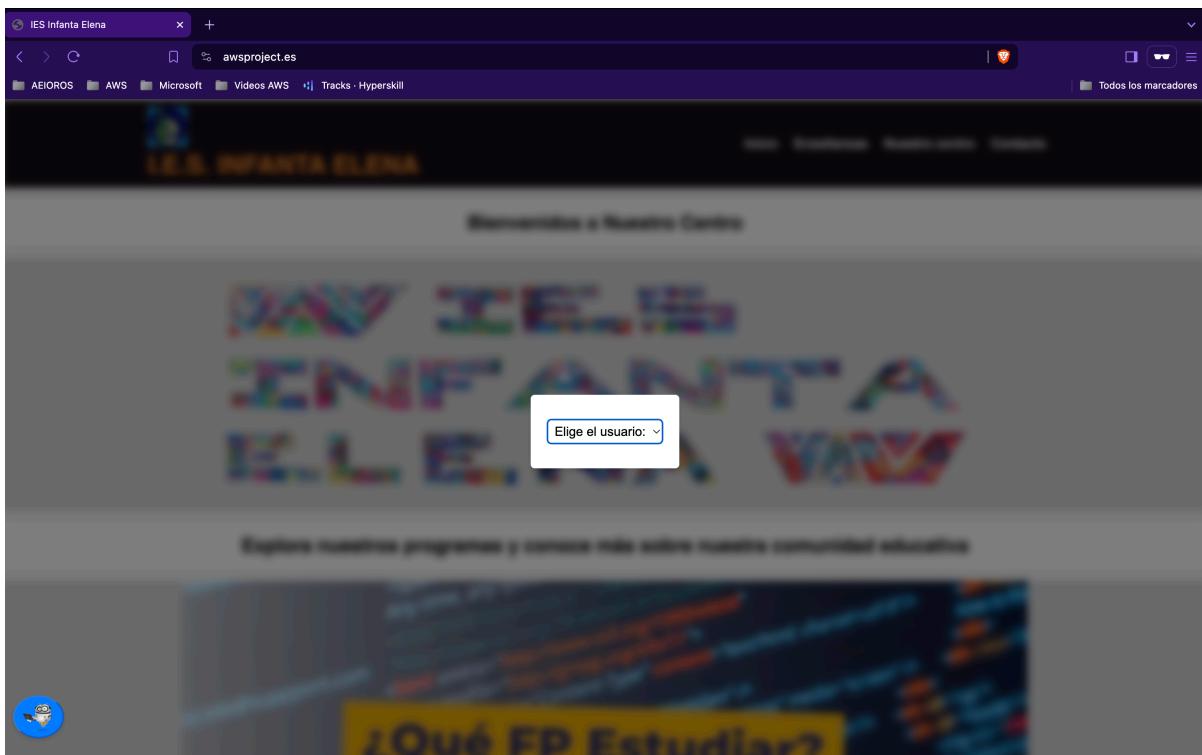
Chrome:



Safari:

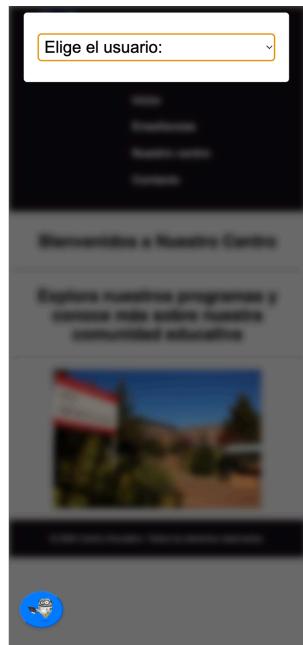


Brave:

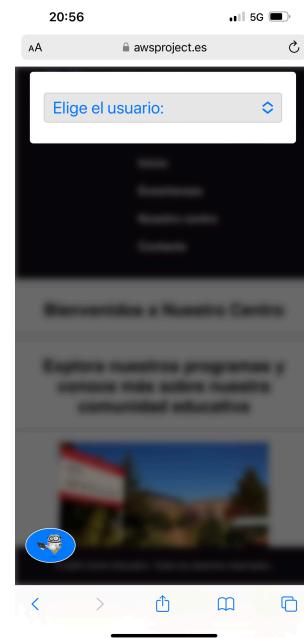


Pruebas en Dispositivos Móviles: Verificar el rendimiento y la presentación del chatbot en dispositivos móviles.

Android Samsung Galaxy A51:

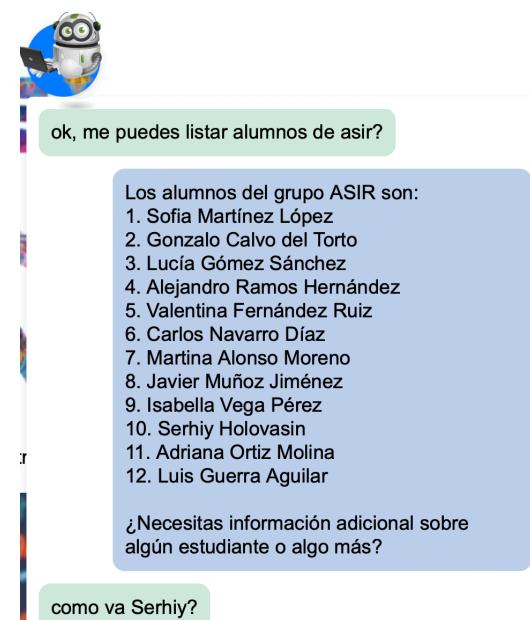


iOS Iphone 13 Pro Max



- *Pruebas Funcionales:*

Consulta de Información Académica: Verificar que el chatbot proporciona información correcta sobre grupos, horarios, asignaturas, y notas.



Interacción con Usuarios: Comprobar que el chatbot gestiona adecuadamente las sesiones de usuarios y mantiene el contexto de las conversaciones.

Hola! Padres de Gonzalo Calvo del Torto

Como estudia mi hijo?

Tu hijo, Gonzalo Calvo del Torto, ha obtenido las siguientes calificaciones en sus asignaturas de Informática:

- Programación: 7, 8, 10, 10
- Inteligencia Artificial: 8, 9, 9, 10, 4, 7, 9
- Redes: 6, 4, 5, 5
- Bases de Datos: 7, 9, 5
- Seguridad Informática: 5, 5, 9
- Desarrollo Web: 7, 4, 10, 5, 5, 4

El promedio de calificaciones de Gonzalo es de aproximadamente 7.13.

Y que asignatura necesita mejorar para subir la nota media?

Para que Gonzalo pueda subir su nota media, es importante que mejore en aquellas asignaturas donde ha obtenido calificaciones más bajas. En este caso, las asignaturas en las que necesita mejorar son:

1. Desarrollo Web: Gonzalo ha obtenido calificaciones de 4, 10, 5, 5, 4 en esta asignatura. Mejorar su rendimiento en Desarrollo Web podría ayudar a elevar su nota media, ya que ha tenido resultados variados en esta área.
2. Redes: Con calificaciones de 6, 4, 5, 5 en Redes, Gonzalo también tiene margen de mejora en esta asignatura para aumentar su nota promedio.
3. Seguridad Informática: Aunque ha obtenido una calificación de 5, 5, 9 en Seguridad Informática, trabajar en mejorar su rendimiento en esta asignatura contribuiría a una posible mejora en su promedio general.

Centrarse en estas asignaturas y dedicar tiempo adicional a comprender los conceptos y practicar activamente puede ser beneficioso para Gonzalo en su objetivo de elevar su calificación media.

- *Pruebas de Usabilidad:*

Facilidad de Uso: Evaluar si los usuarios pueden interactuar intuitivamente con el chatbot sin necesidad de instrucciones adicionales.

que mas me puedes proporcionar sobre mi hijo

Para obtener más información detallada sobre el progreso académico y el rendimiento de Gonzalo, sería útil revisar su horario de clases y las fechas de sus evaluaciones. Esto nos permitirá tener una visión más completa de su desempeño en cada asignatura y identificar posibles áreas de mejora.

¿Te gustaría que consulte el horario de clases de Gonzalo y las fechas de sus evaluaciones para proporcionarte más detalles sobre su progreso académico?

si

He obtenido información adicional sobre Gonzalo Calvo del Torto. Aquí está su horario de clases y las fechas de sus evaluaciones próximas:

Horario de Clases:

1. **05/03/2024**
- Programación: 15:15 - 17:00
- Bases de Datos: 17:20 - 19:00
- Seguridad Informática: 19:10 - 20:45
2. **06/03/2024**
- Desarrollo Web: 15:15 - 17:00
- Inteligencia Artificial: 17:20 - 19:00
- Redes: 19:10 - 20:45
3. **07/03/2024**
- Sistemas Operativos: 15:15 - 17:00
- Programación: 17:20 - 19:00
- Bases de Datos: 19:10 - 20:45

Escribe un mensaje... Enviar

Respuesta a Entradas Incorrectas: Pruebas para verificar cómo el chatbot maneja las entradas erróneas o fuera de contexto.

y me puedes listar los alumnos del grupo LOBOS?

En el grupo "LOBOS" no se encontraron alumnos. Los alumnos disponibles pertenecen al grupo "ASIR", "DAW" y "DAM". Si necesitas más información o tienes alguna otra consulta, no dudes en decírmelo.

y el grupo existe?

El grupo "LOBOS" no parece existir en la base de datos de alumnos proporcionada. Parece que los grupos disponibles son "ASIR", "DAW" y "DAM". Si necesitas más información o tienes alguna otra pregunta, por favor házmela saber.

como lo puedo comprobar?

Para verificar si el grupo "LOBOS" realmente existe en la base de datos de la universidad, puedes comunicarte con la administración de la universidad a través de los siguientes medios de contacto:

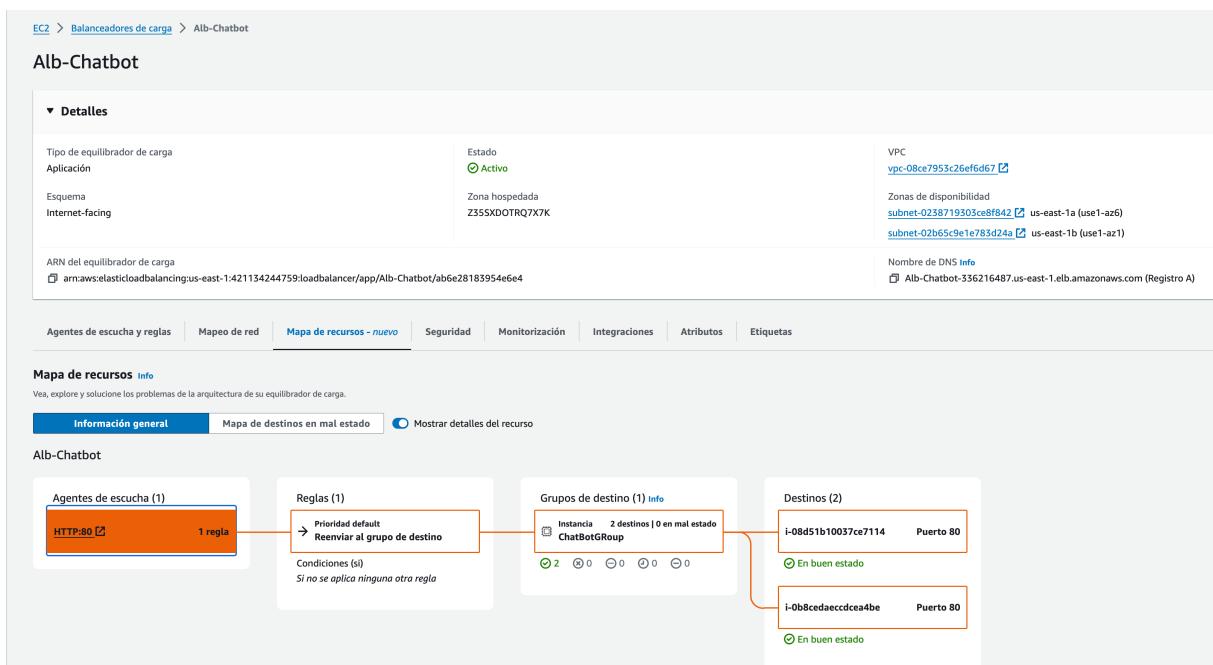
- Teléfono: +34 918 242 342 (disponible de 9:00 a 14:00)
- Email: infanta.elena@educa.madrid.com

Puedes utilizar estos contactos para consultar sobre la existencia del grupo "LOBOS" y cualquier otra información que necesites verificar.

- *Pruebas de Rendimiento*

Pruebas de Escalado de Servicio y Balanceo de Carga:

El chatbot está configurado para operar sobre una arquitectura robusta en AWS, aprovechando la escalabilidad y confiabilidad de los servicios en la nube. Con dos servicios corriendo en distintas zonas de disponibilidad, el sistema está diseñado para asegurar una alta disponibilidad y resistencia frente a posibles fallos.



Esquema 8: Configuración y mapa de recursos de Application Load Balancer

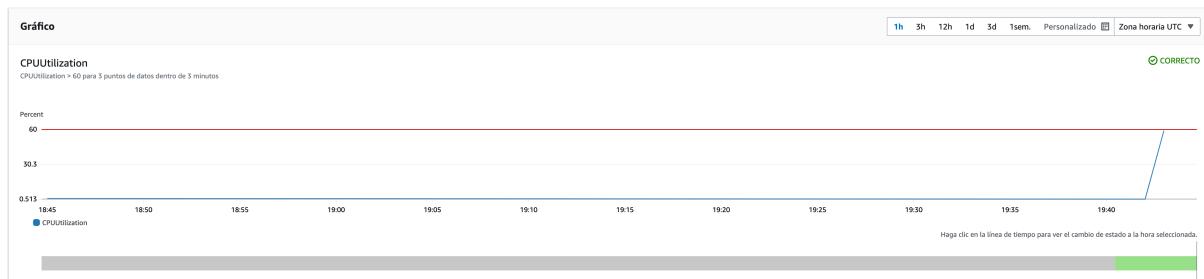
Las políticas de autoescalado están definidas para mantener un mínimo de dos instancias y un máximo de seis, lo que permite al sistema adaptarse dinámicamente a las variaciones en la carga de trabajo.

| CloudWatch > Alarms | | | | | |
|---|------------------|---------------------------------------|--|--------------------------------|--|
| Alarms (2) | | Estado de la alarma: cualq... | Tipo de alarma: cualquiera | Estado de las acciones: cua... | Ocultar alarmas de Auto Scaling Borrar selección |
| Nombre | Estado | Última actualización del estado (UTC) | Condiciones | Acciones | |
| TargetTracking-ChatBot Autoscaling Group-AlarmLow-b1b8b38f-3429-4fc-8d50-f634499ad9e2 | ⚠ En modo alarma | 2024-05-26 17:24:35 | CPUUtilization < 42 para 15 puntos de datos dentro de 15 minutos | acciones habilitadas | |
| TargetTracking-ChatBot Autoscaling Group-AlarmHigh-4e36b05f-3cda-410f-9145-a8106a038ad2 | CORRECTO | 2024-05-26 17:09:11 | CPUUtilization > 60 para 5 puntos de datos dentro de 3 minutos | acciones habilitadas | |

Esquema 9: Configuración de las políticas de autoescalado

La configuración de autoescalado utiliza métricas de CPU como criterios clave para ajustar la cantidad de instancias. Si el uso de CPU supera el 60% durante un período sostenido de tres minutos, el sistema automáticamente inicia una nueva instancia hasta alcanzar un máximo de seis. De manera inversa, si el uso de CPU cae por debajo del 48% y se mantiene así durante 15 minutos, el sistema comienza a reducir el número de instancias activas, pero nunca por debajo de dos. Esta configuración es importante para mantener un balance entre el rendimiento y el coste operativo.

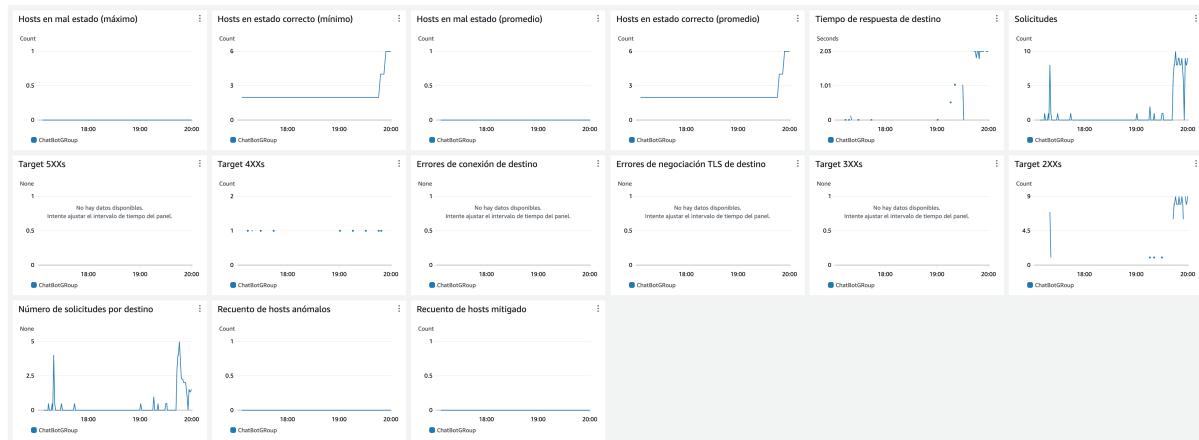
Para probar esta funcionalidad, utilizamos una aplicación especializada en generar tráfico intenso, capaz de simular condiciones de uso máximo y llevar el uso de CPU al 100%.



Esquema 10: Métricas de CPU en utilización en el momento.

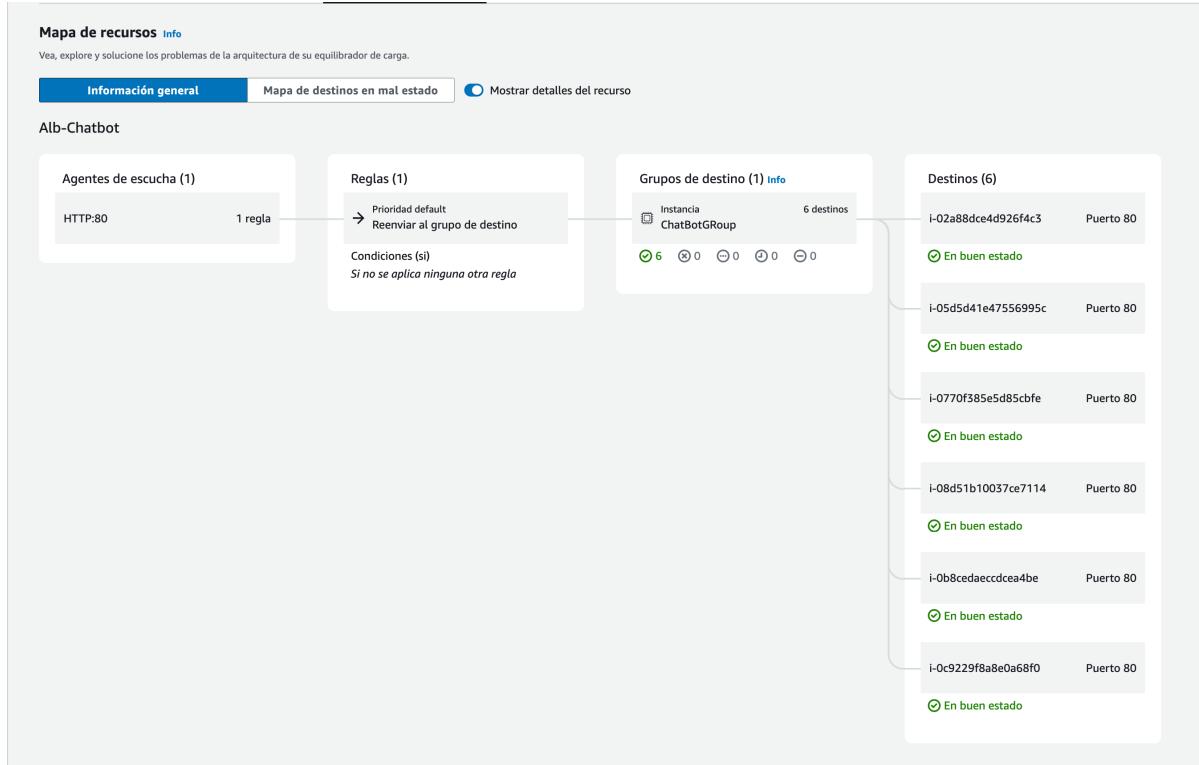
Durante las pruebas, monitoreamos continuamente el sistema utilizando Amazon CloudWatch, lo que nos permite observar en tiempo real cómo se ajustan las instancias a las fluctuaciones en la demanda. Registramos todos los eventos relevantes para analizar posteriormente la capacidad del sistema de responder a las políticas de autoescalado establecidas.

El análisis de los resultados de estas pruebas es fundamental para validar la eficacia de las políticas de autoescalado. Observamos cómo el sistema responde a incrementos de la carga, activando nuevas instancias según lo necesario y reduciéndolas cuando la demanda decrecía.



Esquema 11: Métricas observadas en CloudWatch.

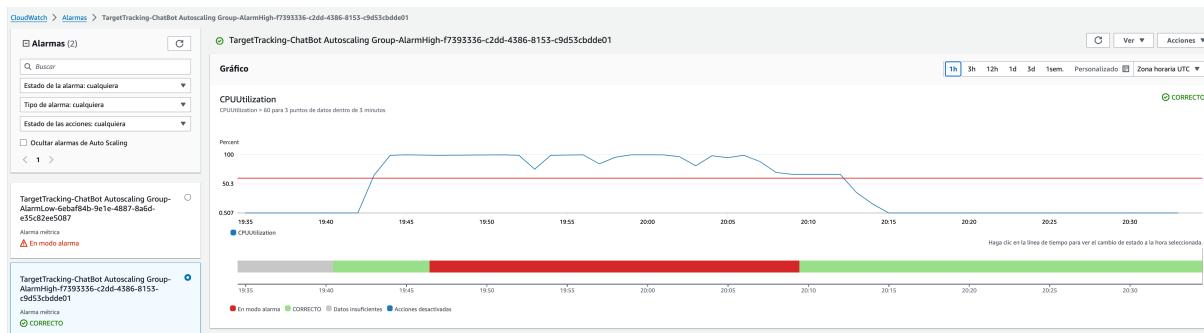
Estos datos nos permiten ajustar las métricas de disparo para optimizar tanto el rendimiento como el costo, asegurando que el chatbot educativo pueda manejar eficientemente picos de demanda sin degradar la experiencia del usuario.



Esquema 12: Mapa de recursos de Application Load Balancer con autoescalado.

La realización de pruebas de escalabilidad y balanceo de carga nos ha proporcionado confianza en la capacidad del sistema de mantener un rendimiento óptimo bajo diversas condiciones de uso. A través de este proceso, hemos asegurado que nuestro chatbot educativo

sea no solo funcional sino también eficiente y capaz de adaptarse a un entorno dinámico, lo cual es importante para su operación continua y su éxito a largo plazo.



Esquema 13: Métricas de CPU en utilización en el momento.

- *Pruebas de seguridad:*

Durante la fase de pruebas de seguridad de nuestro ChatBot Educativo, identificamos un problema crítico relacionado con el manejo de sesiones. Observamos que cuando múltiples usuarios se conectaban simultáneamente al sistema, se producían interferencias en los datos de sesión, resultando en una mezcla y confusión de información entre usuarios. Este descubrimiento fue importante, ya que comprometía tanto la seguridad como la integridad de los datos del usuario, elementos fundamentales para la confiabilidad de nuestra plataforma educativa.

Para abordar este problema, decidimos implementar la utilización de WebSockets. Esta tecnología nos permite establecer un canal de comunicación persistente y bidireccional entre el usuario y el servidor, lo que significa que cada sesión de usuario puede mantenerse aislada y gestionarse de manera independiente. La implementación de WebSockets no solo resolvió el problema de interferencia entre sesiones, sino que también mejoró la interactividad y la capacidad de respuesta del chatbot, asegurando que los mensajes y datos de cada usuario permanezcan privados y seguros durante toda la interacción con el sistema.

Esta solución no solo añadió una capa más de seguridad a nuestra aplicación, sino que también elevó la experiencia del usuario, haciendo que las interacciones con el ChatBot fueran más fluidas y confiables. Las pruebas posteriores confirmaron la efectividad de esta solución, mostrando que los problemas de interferencia de sesiones habían sido completamente mitigados.

```

153     function connectWebSocket(role, name) {
154         const params = new URLSearchParams({ role, name });
155         const url = `ws://${host_url}/chat/ws?${params.toString()}`;
156         socket = new WebSocket(url);
157
158         socket.onopen = function() {
159             console.log('Conectado al WebSocket');
160         };
161
162         socket.onmessage = function(event) {
163             const data = JSON.parse(event.data);
164             displayMessage(data.text, 'bot');
165         };
166
167         socket.onerror = function(error) {
168             console.error('WebSocket Error:', error);
169             displayMessage('Error de conexión WebSocket', 'bot');
170         };
171
172         socket.onclose = function(event) {
173             if (event.wasClean) {
174                 console.log('Conexión cerrada limpiamente');
175             } else {
176                 console.error('Conexión terminada inesperadamente');
177             }
178             displayMessage('Conexión WebSocket cerrada', 'bot');
179         };
180     }
181
182     function sendMessageToBackend(message) {
183         if (socket && socket.readyState === WebSocket.OPEN) {
184             socket.send(JSON.stringify({ text: message }));
185         } else {
186             displayMessage('No se pudo enviar el mensaje. Conexión WebSocket no disponible.', 'bot');
187         }
188     }
189 }
```

Esquema 14: Implementación de WebSockets.

- *Prueba de Funcionamiento del Chatbot*

Con esta prueba pretendemos validar la operatividad y rendimiento del chatbot en condiciones de producción, asegurando que todas las partes integradas del sistema (ALB, ECS, Fargate, Docker con la aplicación del chatbot) funcionen adecuadamente en conjunto.

Para eso hemos implementado una prueba inicial verificando la respuesta del endpoint raíz de nuestra aplicación.

```

EXPLORADOR      ...
PROYECTOFINAL   ...
python_apps
  chatbot
    __init__.py
    .env
    chat_processors.py
    data_processors.py
  models_sql
    __init__.py

root_and_service.py
...
python_apps > routes > root_and_service.py > ...
1   from fastapi import APIRouter
2
3   router = APIRouter(
4       tags=["root"],
5       responses={404: {"description": "Not found"}}
6   )
7
8
9   # Endpoint raíz para verificar que la API está funcionando
10  @router.get("/")
11  def read_root():
12      return {"Hello": "Sergio"} 
```

Esta prueba consiste en acceder a la API a través de la URL del Application Load Balancer (ALB) que distribuye el tráfico a nuestro servicio en ECS y Fargate, donde se ejecuta el chatbot. El código en FastAPI configura un endpoint raíz ("/") que, al ser solicitado, debe retornar un mensaje JSON {"Hello": "Sergio"}. Esta respuesta simple confirma que la aplicación está activa y puede manejar las solicitudes correctamente.



Esta prueba simple proporciona una verificación inicial del funcionamiento del sistema en un entorno real y ayuda a asegurar que el chatbot esté preparado para interactuar con los usuarios finales.

4.3. CONCLUSIONES DE LA FASE DE PRUEBAS

La fase de pruebas de nuestra app ha sido fundamental para asegurar que el sistema no solo cumple con los requisitos técnicos y funcionales especificados, sino que también ofrece una experiencia de usuario óptima. Durante esta fase, hemos validado varios aspectos importantes del sistema, desde la funcionalidad hasta la seguridad, pasando por la usabilidad y la compatibilidad.

1. Cumplimiento de Requisitos Funcionales:

La mayoría de los requisitos funcionales se han cumplido exitosamente. El sistema ha demostrado su capacidad para manejar consultas específicas de diferentes tipos de usuarios (alumnos, profesores, padres) de manera eficiente y segura. Sin embargo, hemos identificado algunas áreas donde la respuesta del chatbot podría ser optimizada, especialmente en el manejo de consultas ambiguas o poco comunes.

2. Seguridad y Gestión de Sesiones:

Uno de los mayores retos identificados fue la gestión de sesiones simultáneas, donde inicialmente se observó una mezcla de datos de sesión entre usuarios. La implementación de WebSockets como solución a este problema ha fortalecido significativamente la seguridad y la integridad de las sesiones de usuario.

3. Usabilidad y Diseño:

Las pruebas de usabilidad indicaron que la interfaz es generalmente fácil de usar y entender. Sin embargo, descubrimos que podríamos hacer algunos cambios para que sea aún más fácil de personalizar según el usuario que la esté usando, ya sea un alumno, un profesor o un parent. Estos resultados nos señalan que debemos trabajar en hacer que la interacción con el chatbot sea más adaptable a las necesidades específicas de cada grupo de usuarios.

4. Utilidad y Impacto Educativo:

En cuanto a la utilidad, el chatbot ha demostrado ser una herramienta valiosa dentro del entorno educativo, proporcionando respuestas rápidas y fiables a las consultas. Aunque ha sido útil, esta aplicación está todavía en sus etapas iniciales. Para mejorar su funcionalidad y expandir su impacto educativo, es necesario desarrollar y añadir más funciones que permitan una interacción más profunda y variada. Además, integrar el chatbot con una base de datos real mejorará significativamente su capacidad para proporcionar información relevante y actualizada, haciendo que la herramienta sea aún más efectiva y práctica para usuarios finales como estudiantes, profesores y padres.

Todas estas pruebas han sido indispensables para identificar y abordar los problemas prácticos del sistema, proporcionando un camino claro para futuras mejoras. Aunque se han cubierto satisfactoriamente la mayoría de los requisitos, el continuo refinamiento y expansión del sistema serán necesarios para mantener su relevancia y efectividad en un entorno educativo dinámico.

5. RESULTADO FINAL

Al culminar el desarrollo de nuestro ChatBot en AWS, hemos alcanzado un momento decisivo para evaluar cómo se han cumplido los objetivos iniciales del proyecto. En esta fase vamos a identificar tanto los logros obtenidos como las áreas que requieren mejoras adicionales, teniendo en cuenta las restricciones de tiempo y los recursos limitados del entorno de laboratorio gratuito.

Evaluación de Objetivos:

1. *Objetivo de Mejorar la Comunicación en Centros Educativos:*

El objetivo principal era mejorar la comunicación entre estudiantes, profesores y padres mediante un sistema automatizado. Hemos logrado desarrollar un ChatBot que facilita interacciones inmediatas para consultas comunes y proporciona un nivel significativo de soporte automatizado. Aunque el sistema ha demostrado ser funcional en un contexto de prueba, es importante reconocer que este proyecto representa solo el comienzo de lo que podría ser una implementación más amplia. La integración real y completa en el centro educativo ofrecería la oportunidad de afinar aún más el sistema, ajustando y mejorando sus capacidades en función de las necesidades y feedback directo de los usuarios finales.

El chatbot, apoyado por la tecnología de AWS y la API de OpenAI, ha demostrado ser una herramienta efectiva para proporcionar comunicación instantánea y continua, cumpliendo con las expectativas de inmediatez requeridas por un entorno educativo moderno. Aunque el sistema fue probado en un entorno controlado, demostró la capacidad de operar ininterrumpidamente, ilustrando los beneficios de la escalabilidad y alta disponibilidad que AWS facilita. Sin embargo, dado que este es un proyecto dentro de un laboratorio gratuito con recursos y tiempo limitados, aún queda mucho trabajo por hacer para optimizar y personalizar el sistema completamente.

2. *Objetivo de Disponibilidad y Escalabilidad:*

El sistema fue diseñado para ser escalable y asegurar disponibilidad continua, utilizando la infraestructura de AWS para funcionar las 24 horas del día, 7 días a la semana. Aunque el chatbot ha gestionado un volumen básico de consultas efectivamente y sin fallos de rendimiento significativos, hemos identificado una necesidad de fortalecer la infraestructura para manejar de manera óptima los picos de tráfico durante períodos de alta demanda.

En cuanto a la protección de la información y la privacidad, el uso de la tecnología de OpenAI ha hecho que nuestro chatbot sea más inteligente y eficaz al responder preguntas. Sin embargo, es muy importante que sigamos trabajando para hacer más seguros el sistema y los datos de los usuarios. Esto significa que debemos estar siempre atentos y aplicar medidas más estrictas para asegurarnos de que los datos personales estén bien cuidados y a salvo de cualquier acceso o uso no autorizado.

3. Reducción de la Carga Administrativa:

La automatización de respuestas a consultas frecuentes ha demostrado la capacidad del chatbot para aliviar la carga administrativa del personal educativo. Sin embargo, con limitaciones de tiempo y en el alcance de las funcionalidades implementadas hasta la fecha, queda claro que el pleno potencial del sistema aún está por realizarse. Además, hemos comenzado a ver cómo el chatbot puede reducir la cantidad de trabajo repetitivo que enfrenta el personal educativo.

4. Objetivo de Usabilidad y Personalización:

En cuanto a la usabilidad y experiencia del usuario, el chatbot ha sido bien recibido gracias a su interfaz intuitiva y fácil de manejar. Este está diseñado para ser adaptable y flexible, lo que le permite funcionar eficazmente en una variedad de entornos educativos. Aunque el sistema puede adaptar sus respuestas y funcionalidades a las necesidades específicas de cada usuario, todavía requiere desarrollo adicional para lograr la versatilidad que buscamos. Además, es necesario realizar más mejoras en su configuración para asegurar una personalización completa que se ajuste a los requisitos particulares de cada centro educativo.

“Este proyecto ha sentado las bases para lo que podría ser una herramienta transformadora en la comunicación educativa, ofreciendo insights valiosos sobre cómo un chatbot puede facilitar la interacción dentro de los centros educativos. Aunque no hemos implementado el sistema en un entorno real, los resultados obtenidos apuntan a un potencial significativo para futuras implementaciones. Hay que seguir desarrollando y mejorando el chatbot, ajustándolo para alcanzar el ideal de que comunicarse con un centro educativo sea tan fácil y accesible como chatear con un amigo. El apoyo continuo de la tecnología de OpenAI será de gran ayuda en este proceso de evolución, garantizando que el sistema no solo sea más inteligente, sino también más seguro y útil para todos los usuarios. Al final, la implementación de este proyecto no solo demuestra la viabilidad de la tecnología de chatbots en la educación, sino que también destaca el compromiso del centro educativo con la innovación y la mejora continua en el servicio a su comunidad.”

6. REFERENCIAS

Chatbots y Estudios Relacionados:

1. Sharoon, E(21 de agosto de 2019). La línea de tiempo: Historia de Chatbots - Antes, ahora y mañana. Planeta ChatBot. Recuperado de <https://planetachatbot.com/linea-tiempo-historia-de-chatbots-antes-ahora-y-manana/>
2. Bartolomé, A., Cruzado Asencio, G., Molins, L., Martínez Monés, A., Martins, L., Paredes Rizo, C., Polo Chávarri, L., Rejón-Guardia, F., Uceda Martos, P., Vich-i-Martorell, G. (2023). Chatbots en Educación. Recuperado de https://www.lmi-cat.net/sites/default/files/Chatbots_en_Educacion.pdf
3. UTEC. (6 de julio de 2023). ¿Qué es un chatbot, para qué sirve y cómo incluirlo en el aula? Recuperado de <https://utec.edu.uy/es/noticia/que-es-un-chatbot-para-que-sirve-y-como-incluirlo-en-el-aula/>
4. Bustamante, P. (21 de enero de 2024). Innovación educativa: uso de chatbots en el aula. Aula Simple. Recuperado de <https://aulasimple.ai/blog/innovacion-educativa-uso-de-chatbots-en-el-aula/>

Python

5. Python Software Foundation. Última actualización el 20 de abril de 2024. Recuperado de <https://docs.python.org/3.13/>
6. Uchenna, E (15 de junio de 2023). Crea una Aplicación con FastAPI para Python. Kinsta. Recuperado de <https://kinsta.com/es/blog/fastapi/>

AWS (Amazon Web Services)

7. Amazon Web Services, Inc. (2024). AWS Documentation. Recuperada de <https://aws.amazon.com/documentation/>

Docker

8. Base de conocimientos(13 de enero de 2023). Qué es Docker: Una Guía Completa . Kinsta. Recuperado de <https://kinsta.com/es/base-de-conocimiento/que-es-docker/>

Imagenes

Imagen 1. Holovasin Suslov S. (2024). “*La clase del futuro*”. Creada con inteligencia artificial mediante DALL·E. [Licencia CC BY-NC 4.0].

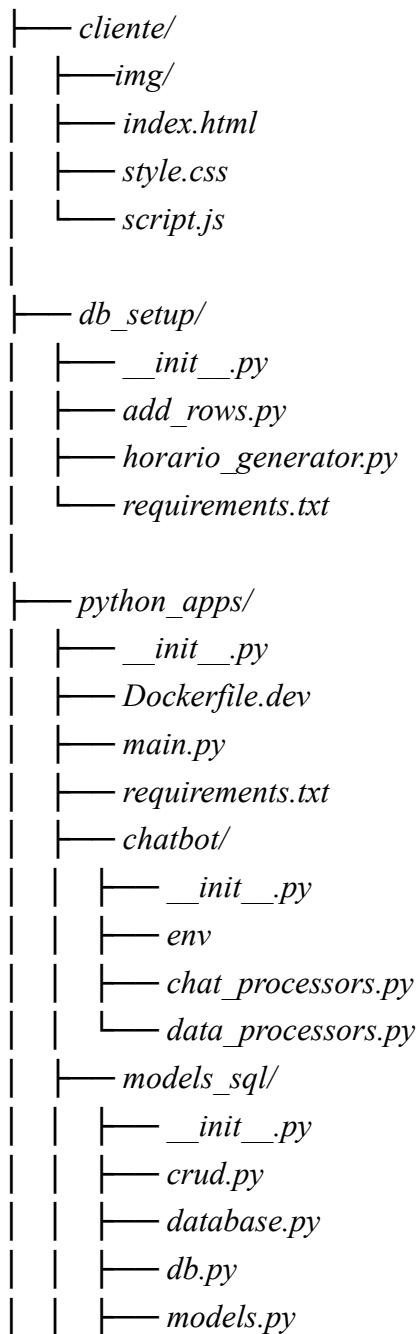
Imagen 2. Holovasin Suslov S. (2024). “*Evolución de la IA y los Chatbots*”. Creada con inteligencia artificial mediante DALL·E. [Licencia CC BY-NC 4.0].

7. ANEXOS

7.1. ANEXO: Estructura del Proyecto. Código.

El proyecto está estructurado en varias carpetas, cada una con un propósito específico: cliente para la interfaz web, db_setup para la configuración de la base de datos, y python_apps para la lógica del chatbot y la API.

ProyectoFinal/



```
    |   └── schemas.py
    └── routes/
        ├── __init__.py
        ├── alumnos.py
        ├── asignaturas.py
        ├── chat.py
        ├── grupos.py
        ├── horarios.py
        ├── horas.py
        ├── notas.py
        ├── profesors.py
        ├── root_and_service.py
        └── temarios.py
    └── sql_app.db
```

7.2. ANEXO: Código del Chatbot: Parte del Cliente

A continuación se presentan las conclusiones clave junto con una explicación detallada del código JavaScript que gestiona la interactividad del chatbot.

Interactividad y Usabilidad:

- El JavaScript maneja la interactividad, permitiendo a los usuarios enviar mensajes y recibir respuestas en tiempo real.
- La funcionalidad de un botón de chat desplegable facilita el acceso rápido al chatbot, mejorando la accesibilidad y usabilidad.

Flujo de Mensajes:

- El código JavaScript se encarga de capturar el mensaje del usuario, enviarlo al servidor y mostrar la respuesta del chatbot en la interfaz de chat.
- Esto crea una experiencia de usuario fluida y dinámica, esencial para mantener el compromiso de los usuarios.

index.html:

```
<div id="chatbot-container">
  <button id="chat-toggle" onclick="toggleChatbot()">
    |   
  </button>
  <div id="chat-box" class="chat-closed">
    |   <div class="messages"></div>
    |   <div style="display: flex; align-items: center;">
    |     |   <textarea id="chat-input" placeholder="Escribe un mensaje..." rows="1"></textarea>
    |     |   <button id="send-btn">Enviar</button>
    |   </div>
  </div>
</div>
```

Script.js:

Function toggleChatbot(): Esta función alterna la visibilidad del chatbot agregando o removiendo la clase open al contenedor del chat. Esto permite mostrar u ocultar el chat cuando el usuario hace clic en la imagen de chatbot correspondiente.

Envío de Mensajes:

```

4  // Función para mostrar/ocultar el chatbot
5  function toggleChatbot() {
6      var chatBox = document.getElementById('chat-box');
7      // Alterna la visibilidad del chatbox
8      if (chatBox.style.display === 'none') {
9          chatBox.style.display = 'block';
10         // Solo muestra el mensaje de bienvenida si aún no se ha mostrado
11         if (!welcomeMessageShown) {
12             displayMessage("¡Hola! ¿Cómo puedo ayudarte?", 'bot');
13             welcomeMessageShown = true; } // Marca que el mensaje ya ha sido mostrado
14     } else {
15         chatBox.style.display = 'none';
16     }
17 }
```

`Document.getElementById('send-btn').addEventListener('click', function())`: Este evento escucha el clic en el botón "Enviar". Cuando se hace clic, obtiene el valor del input, verifica que no esté vacío y luego crea un elemento de mensaje del usuario para mostrar en la interfaz.

```

44 // Agrega el evento de clic al botón de enviar
45 document.getElementById('send-btn').addEventListener('click', sendBtnClick);
46
47 // Función para mostrar mensajes en la interfaz de chat
48 function displayMessage(message, sender) {
49     const messagesContainer = document.querySelector('.messages');
50     const messageDiv = document.createElement('div');
51     messageDiv.classList.add('message');
52     // Aplica clase según el remitente del mensaje
53     if (sender === 'user') {
54         messageDiv.classList.add('user-message');
55     } else {
56         messageDiv.classList.add('bot-message');
57     }
58     messageDiv.innerHTML = message.replace(/\n/g, '  
'); // Establece el texto del mensaje
59     messagesContainer.appendChild(messageDiv); // Añade el mensaje al contenedor
60     messagesContainer.scrollTop = messagesContainer.scrollHeight; // Desplaza al último mensaje
61 }
62 }
```

Comunicación con el Backend:

- `fetch('/chat')`: Esta parte del código envía el mensaje del usuario al backend utilizando una solicitud POST. El mensaje se envía en formato JSON.
- `.then(response => response.json()).then(data =>)`: Una vez que el backend responde, este código crea un elemento de mensaje del bot y lo agrega al contenedor de mensajes para mostrar la respuesta del chatbot.
- `.catch(error => console.error('Error:', error))`: Esta parte del código maneja cualquier error que ocurra durante la comunicación con el backend, asegurando que los errores se registren en la consola para su depuración.

```
// Función para enviar mensajes al servidor y manejar la respuesta
function sendMessageToBackend(message) {
    fetch('http://127.0.0.1:8095/chat/', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json', // Especifica el tipo de contenido
        },
        body: JSON.stringify({ message: message }), // Procesa la respuesta en formato JSON
    })
    .then(response => response.json())
    .then(data => {
        displayMessage(data.response, 'bot'); // Muestra la respuesta del bot
    })
    .catch(error => {
        console.error('Error:', error); // Maneja errores de conexión o del servidor
        displayMessage('Error al enviar mensaje', 'bot'); // Notifica al usuario el error
    });
}
```

La implementación de la interactividad del chatbot a través de JavaScript es crucial para proporcionar una experiencia de usuario dinámica y efectiva. El código permite que los usuarios envíen mensajes y reciban respuestas en tiempo real, mejorando significativamente la comunicación entre los estudiantes y el centro educativo. Esta funcionalidad no solo facilita la interacción, sino que también optimiza la accesibilidad y usabilidad del chatbot educativo.

7.3. ANEXO: Configuración del Backend en python_apps

Esta sección del anexo describe la configuración y estructura del backend del proyecto de chatbot educativo. Se centra en cómo se utilizan Python y tecnologías asociadas para crear un entorno de backend robusto y escalable.

El backend del chatbot se implementa en Python, utilizando el framework FastAPI para manejar las solicitudes HTTP de manera eficiente. Además, se usa Docker para asegurar que la aplicación pueda ser desplegada en cualquier entorno con facilidad y consistencia.

Componentes Clave:

- *main.py*: Este archivo es el punto de entrada de la aplicación FastAPI, también configura las rutas y dependencias del servidor web.

```
python_apps > main.py > configure_app
1  # Importaciones necesarias para la API, manejo de CORS, validación de datos, manejo de variables de entorno y la API de OpenAI
2  from fastapi import FastAPI, HTTPException, Depends
3  from fastapi.middleware.cors import CORSMiddleware
4  from models_sql.db import create_all_models
5
6  from python_apps.routes import grupos, alumnos, root_and_service, asignaturas, profesores, horas, horarios, notas, temarios, chat
7
8  # Inicializa la aplicación FastAPI
9  app = FastAPI()
10
11
12 def configure_app():
13     # Lista de las direcciones de origen que están permitidas para realizar solicitudes CORS a mi API
14     origins = [
15         "http://127.0.0.1:8095"
16     ]
17
```

app = FastAPI(): Inicializa la aplicación FastAPI.

```
18     # Configura CORS para permitir todas las orígenes, métodos y headers
19     # En producción, debemos restringir esto a orígenes confiables
20     app.add_middleware(
21         CORSMiddleware,
22         allow_origins=["*"], # Permite acceso desde cualquier origen
23         allow_credentials=True,
24         allow_methods=["*"], # Permite todos los métodos
25         allow_headers=["*"]) # Permite todos los headers
26
27     app.include_router(root_and_service.router)
28     app.include_router(alumnos.router)
29     app.include_router(asignaturas.router)
30     app.include_router(grupos.router)
31     app.include_router(horas.router)
32     app.include_router(horarios.router)
33     app.include_router(notas.router)
34     app.include_router(profesores.router)
35     app.include_router(temarios.router)
36     app.include_router(chat.router)
```

app.include_router(): Incluye las rutas definidas en otros módulos, organizando las diferentes partes de la API.

- *Dockerfile.dev*: Define la configuración del entorno Docker para desarrollo y especifica la imagen base de Python, las dependencias necesarias y los comandos para ejecutar la aplicación. (*Esta por definir todavía*)
- *__init__.py*: Archivo utilizado para convertir directorios en paquetes Python, permitiendo que otros archivos dentro del mismo directorio sean importados como módulos.
- *requirements.txt*:

Lista las dependencias necesarias para el proyecto.

```
python_apps > └── requirements.txt
1   fastapi==0.110.0
2   pydantic
3   openai
4   uvicorn
5   python-dotenv==1.0.1
6   sqlalchemy
7   tiktoken
```

Esta configuración del backend es importante para soportar la interactividad del chatbot y manejar efectivamente las solicitudes de los usuarios, asegurando que el sistema sea robusto, escalable y fácil de mantener.

7.4. ANEXO : Modelos y Rutas de la API en python_apps

Aquí podemos observar la configuración de los modelos de base de datos y las rutas de la API que manejan diferentes entidades del sistema del chatbot educativo. Nos centraremos en la funcionalidad clave y el propósito de cada componente.

Las rutas de la API están organizadas en la carpeta *routes*, y cada archivo dentro de esta carpeta maneja una entidad específica, como alumnos, profesores, notas, horarios, entre otros. Estas rutas utilizan FastAPI para definir los endpoints que interactúan con la base de datos y procesan las solicitudes del cliente.

- *alumnos.py*: Maneja las rutas relacionadas con los alumnos.

```
python_apps > routes > ✎ alumnos.py > ...
  1  from fastapi import APIRouter, Depends
  2  from sqlalchemy.orm import Session
  3  from ..models_sql import crud, schemas, db, models
  4
  5  # Configuración inicial de rutas y modelos
  6  subprefix = "alumno"
  7  db_model = models.Alumno
  8  db_schema = schemas.Alumno
  9
 10 # Creación del enrutador para el módulo de alumnos
 11 router = APIRouter(
 12     prefix="/alumnos",      # Prefijo común para todas las rutas
 13     tags=[subprefix],       # Etiqueta para agrupar todos los endpoints de a
 14     responses={404: {"description": "The object does not exist"}}
 15 )
 16
 17 # Endpoint para crear un nuevo alumno
 18 @router.post("/create")
 19 def alumno_create(obj: schemas.Alumno, db: Session = Depends(db.get_db)):
 20     return crud.alumno_create(obj=obj, db=db)
 21
 22 # Endpoint para buscar todos los alumnos
 23 @router.get("/search")
 24 def search(db: Session = Depends(db.get_db)):
 25     return db.query(db_model).all()
 26
 27 # Endpoint para obtener un alumno específico por ID
 28 @router.get(f"/{subprefix}/{obj_id}", response_model=db_schema)
 29 def get(obj_id: int, db: Session = Depends(db.get_db)):
 30     return crud.get_object(obj_id=obj_id, db=db, model=db_model)
 31
 32 # Endpoint para eliminar un alumno específico por ID
 33 @router.delete(f"/{subprefix}/{obj_id}")
 34 def delete(obj_id: int, db: Session = Depends(db.get_db)):
 35     return crud.delete_object(obj_id=obj_id, db=db, model=db_model)
```

Define las rutas para CRUD operaciones (Create, Read, Update, Delete) sobre los alumnos y Usa dependencias de FastAPI para obtener la sesión de la base de datos.

- *profesores.py*: Define los endpoints para gestionar la información de los profesores.
- *asignaturas.py*: Maneja las rutas relacionadas con las asignaturas.
- *grupos.py*: Define los endpoints para gestionar los grupos de alumnos.
- *horas.py*: Gestiona las rutas para manejar las horas de clases.
- *temarios.py*: Administra las rutas relacionadas con los temarios de las asignaturas.
- *notas.py*: Gestiona las rutas relacionadas con las notas de los alumnos.
- *horarios.py*: Administra los endpoints para consultar y gestionar los horarios.
- *root_and_service.py*: Contiene rutas generales y servicios comunes que no se ajustan a una entidad específica.
- *chat.py*: Define los endpoints para la interacción con el chatbot.

```
python_apps > routes > 🗂️ chat.py > ⚙️ chat
1   from fastapi import APIRouter, Depends, HTTPException
2   from python_apps.chatbot.chat_processors import ChatRequest, generate_response
3
4   router = APIRouter(
5       prefix="/chat",
6       tags=["chat"],
7       responses={404: {"description": "Not found"}}
8   )
9
10  # Endpoint que maneja la lógica de chat, recibiendo un mensaje y devolviendo la respuesta del chat
11  @router.post("/")
12  async def chat(chat_request: ChatRequest):
13      user_input = chat_request.message
14      # Imprime el input para depuración (eliminar o cambiar por logging adecuado en producción)
15      print(user_input)
16
17      # Lanza un error HTTP si el mensaje está vacío
18      if not user_input:
19          raise HTTPException(status_code=400, detail="No message provided")
20
21      # Genera la respuesta del chat y la devuelve
22      chat_response = await generate_response(user_input)
23
24      return {"response": chat_response}
```

Se define una ruta con el prefijo */chat* y una etiqueta *chat* para organizar los endpoints relacionados con el chatbot.

El endpoint / maneja solicitudes POST. Recibe un mensaje del usuario encapsulado en una clase ChatRequest y verifica que el mensaje no esté vacío. Si lo está, lanza una excepción HTTP 400. Utiliza la función *generate_response* para generar una respuesta basada en el mensaje del usuario y retorna la respuesta generada en formato JSON.

La organización de los modelos y rutas de la API en el proyecto asegura que cada entidad tenga su propio conjunto de endpoints claramente definidos. Esto facilita el mantenimiento del código y la ampliación de funcionalidades en el futuro. FastAPI proporciona una manera eficiente de manejar las solicitudes HTTP y de interactuar con la base de datos, asegurando un rendimiento óptimo y una respuesta rápida a las consultas de los usuarios.

7.5. ANEXO: Configuración de la Base de Datos y Modelos en python_apps

Esta sección del anexo describe la configuración de la base de datos y los modelos que manejan diferentes entidades del sistema del chatbot educativo. Nos centraremos en la funcionalidad clave y el propósito de cada componente.

La configuración de la base de datos y los modelos está organizada en la carpeta models_sql. Utiliza SQLAlchemy para definir los modelos de base de datos y Pydantic para la validación de datos. FastAPI facilita la interacción con estos modelos a través de dependencias inyectadas.

- *models.py*: Define los modelos de las tablas de la base de datos y se establecen relaciones entre ellas utilizando *relationship*.

```
python_apps > models_sql > 🗂️ models.py > 📄 Alumno
1  from sqlalchemy import create_engine, Column, Integer, String, ForeignKey, Date
2  from sqlalchemy.ext.declarative import declarative_base
3  from sqlalchemy.orm import relationship, sessionmaker, Mapped
4  from typing import List
5
6  # Define la base declarativa de SQLAlchemy desde la que heredarán todos los modelos de la base de datos
7  Base = declarative_base()
8
9  # Modelos
10
11 class Alumno(Base):
12     __tablename__ = 'alumnos'                                # Nombre de la tabla en la base de datos
13     id = Column(Integer, primary_key=True, autoincrement=True) # Columna de ID, clave primaria autoincrementable
14     nombre = Column(String, unique=True)                      # Nombre del alumno, debe ser único
15     grupo_id = Column(Integer, ForeignKey('grupos.id'))      # Clave foránea que apunta a la tabla de grupos
16     grupo: Mapped["Grupo"] = relationship(back_populates="alumnos") # Relación con la clase Grupo
17
```

- *schemas.py*: Define los esquemas Pydantic para la validación de datos.

```
python_apps > models_sql > 🗂️ schemas.py > 📄 Temario
1  from pydantic import BaseModel
2  from datetime import date
3
4
5  class Alumno(BaseModel):
6      id: int | None = None          # ID opcional del alumno, útil para cuando los objetos aún no están guardados en la base de datos
7      nombre: str                  # Nombre del alumno
8      grupo_id: int                # ID del grupo al que pertenece el alumno
9
10
11     class Config:
12         orm_mode = True
13
14 class Asignatura(BaseModel):
15     id: int | None = None
16     nombre: str
17
18     class Config:
19         orm_mode = True
```

- *database.py*: Configura la conexión a la base de datos, configurando la URL de la base de datos y el motor de SQLAlchemy.

```
python_apps > models_sql > database.py > ...
1  from sqlalchemy import create_engine
2  from sqlalchemy.ext.declarative import declarative_base
3  from sqlalchemy.orm import sessionmaker
4
5  ## Define la URL de conexión a la base de datos usando SQLite
6  SQLALCHEMY_DATABASE_URL = "sqlite:///sql_app.db"
7  # Alternativa de URL para conectar con una base de datos PostgreSQL
8  # SQLALCHEMY_DATABASE_URL = "postgresql://user:password@postgresserver/db"
9
10 # Crea un motor de base de datos, que gestiona las conexiones a la base de datos
11 # Para SQLite, se desactiva la verificación del mismo hilo debido a restricciones de SQLite con Flask.
12 engine = create_engine(
13     SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
14 )
15
16 # Crea una fábrica de sesiones de SQLAlchemy que será la clase base para todas las sesiones de base de datos creadas
17 SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
18 # Crea una clase base para los modelos declarativos, que mantiene un catálogo de clases y tablas mapeadas a la base de datos
19 Base = declarative_base()
```

- *crud.py*: Contiene las operaciones CRUD para los modelos.

```
python_apps > models_sql > crud.py > create_object
1  from sqlalchemy.orm import Session
2  from sqlalchemy.exc import IntegrityError
3  from fastapi import HTTPException
4
5  from . import models, schemas
6
7  # Crear operaciones
8  def create_object(db: Session, model, schema_data):
9      """Crea un objeto en la base de datos según el modelo y los datos del esquema."""
10     new_obj = model(**schema_data.dict())
11     try:
12         db.add(new_obj)
13         db.commit()                      # Guarda el nuevo objeto en la base de datos.
14         db.refresh(new_obj)            # Actualiza la instancia con los datos de la base de datos.
15         return new_obj
16     # Maneja errores de integridad como duplicados o datos faltantes.
17     except IntegrityError as exc:
18         db.rollback()                  # Revierte la transacción si hay un error.
19         raise HTTPException(
20             status_code=400,
21             detail=f"Failed to add new row. Exception: {str(exc)}"
22         )
23
24
25
26 def alumno_create(db: Session, obj: schemas.Alumno):
27     """Crea un objeto alumno si no existe."""
28     obj_exist = db.query(models.Alumno).filter_by(nombre=obj.nombre).first()
29     if obj_exist:
30         return obj_exist
31     return create_object(db, models.Alumno, obj)
32
33 def asignatura_create(db: Session, obj: schemas.Asignatura):
34     """Crea un objeto asignatura si no existe."""
35     obj_exist = db.query(models.Asignatura).filter_by(nombre=obj.nombre).first()
36     if obj_exist:
37         return obj_exist
38     return create_object(db, models.Asignatura, obj)
39
40 def grupo_create(db: Session, obj: schemas.Grupo):
41     """Crea un objeto grupo si no existe."""
42     obj_exist = db.query(models.Grupo).filter_by(nombre=obj.nombre).first()
43     if obj_exist:
44         return obj_exist
45     return create_object(db, models.Grupo, obj)
```

En el contexto de desarrollo de software, CRUD es un acrónimo para las operaciones básicas que se pueden realizar sobre una base de datos o un conjunto de datos específico. Estas operaciones son:

- Create (Crear): Refiere a la capacidad de crear nuevos registros o datos.
- Read (Leer): Se refiere a la capacidad de leer, consultar o ver los datos existentes.
- Update (Actualizar): Implica la capacidad de modificar o actualizar datos existentes.
- Delete (Eliminar): Se refiere a la capacidad de eliminar datos existentes.

- *db.py*: Inicializa la base de datos y crea las tablas.

La configuración de la base de datos y los modelos en el proyecto del chatbot educativo asegura que los datos se almacenen y gestionen de manera eficiente. SQLAlchemy proporciona una manera robusta de definir modelos y relaciones, mientras que Pydantic facilita la validación de datos y la interacción con la API.

```
python_apps > models_sql > db.py > get_db
1
2  from python_apps.models_sql.database import SessionLocal, engine
3
4  def get_db():
5      """
6          Genera una instancia de sesión de base de datos y garantiza su cierre tras el uso.
7
8          Esta función es un generador que se utiliza típicamente en FastAPI para manejar
9          la sesión de la base de datos en un contexto de dependencia. Asegura que la sesión
10         se cierre correctamente después de que la petición se procese."""
11
12     # Crea una nueva sesión de base de datos
13     db = SessionLocal()
14     try:
15         # Proporciona la sesión al contexto que la invoca
16         yield db
17     finally:
18         # Asegura que la sesión se cierre adecuadamente
19         db.close()
20
21
22 def create_all_models():
23     """
24         Crea todas las tablas en la base de datos según los modelos definidos.
25
26         Esta función importa los modelos y utiliza el motor de base de datos para crear
27         todas las tablas definidas en la metadata de SQLAlchemy. Se ejecuta típicamente
28         al inicio del proyecto para asegurar que la base de datos esté preparada para
29         ser utilizada por la aplicación.
30     """
31
32     # Importa la clase base de los modelos
33     from .models import Base
34     # Crea todas las tablas en la base de datos
35     Base.metadata.create_all(bind=engine)
```

7.6. ANEXO: Lógica del Chatbot

En este anexo se describe la lógica del chatbot, centrándonos en los archivos `_init_.py`, `chat_processors.py` y `data_processors.py`, los cuales gestionan el procesamiento de los mensajes y los datos necesarios para el funcionamiento de esta aplicación.

El chatbot está diseñado para interactuar con los usuarios, procesar sus mensajes y generar respuestas adecuadas. Para ello, se utilizan tres módulos principales:

7.1. `__init__.py`

Este archivo configura las importaciones básicas y las definiciones iniciales para la aplicación de chatbot. Establece las herramientas y funciones que se utilizan en el chatbot para interactuar con los usuarios:

- Importación de módulos y funciones necesarias de otros archivos.
- Definición de una lista de herramientas (tools) con sus especificaciones para ser utilizadas en las interacciones del chat.

```
python_apps > chatbot > __init__.py > ...
1  import json
2  from python_apps.chatbot.data_processors import get_professors, get_groups, get_students, get_notas_of_alumno, get_schedule_for_date
3  tools = [
4      {
5          'type': "function",
6          'function': {
7              'name': 'get_professors',
8              'description': 'Obtener lista de profesores del centro educativo con sus asignaturas',
9              'parameters': {
10                  'type': 'object',
11                  'properties': {
12                      'subject': {
13                          'type': 'string',
14                          'description': 'La asignatura que enseña el profesor. Utilizado como filtro en la base de datos'
15                      }
16                  }
17              }
18          }
19      },
20      {
21          'type': "function",
22          'function': {
23              'name': 'get_students',
24              'description': 'Obtener lista de estudiantes de la universidad',
25              'parameters': {
26                  'type': 'object',
27                  'properties': {
28                      'group': {
29                          'type': 'string',
30                          'description': 'El grupo al que pertenece el estudiante.'
31                      }
32                  }
33              }
34          }
35      }
36  ]
```

- Mapeo de nombres de funciones a las implementaciones reales, permitiendo su llamado dinámico durante la operación del chatbot.

```

102     # Prompt inicial que configura el rol y contenido de las interacciones iniciales
103     initial_prompt = {
104         "role": "system",
105         "content": prompt_instructions
106     }

```

- Prompt inicial que configura el rol y contenido de las interacciones iniciales

```

82
83     # Diccionario que mapea nombres de funciones a las funciones reales importadas
84     functions = {
85         'get_professors': get_professors,
86         'get_groups': get_groups,
87         'get_students': get_students,
88         'get_grades': get_notas_of_alumno,
89         'get_schedule_for_date': get_schedule_for_date
90     }
91

```

7.2. *chat_processors.py*

Este archivo contiene la lógica para procesar las interacciones de los usuarios mediante el chatbot, utilizando el modelo de OpenAI y gestionando el historial de la conversación:

```

python_apps > chatbot > 🗃 chat_processors.py > ...
1  from dotenv import load_dotenv
2  from pydantic import BaseModel
3  from fastapi import HTTPException
4  import openai
5  import os
6  from python_apps.main import app
7  from python_apps.chatbot.data_processors import get_professors
8  from python_apps.chatbot import initial_prompt, tools, functions
9
10 # Carga las variables de entorno desde el archivo .env
11 load_dotenv()
12
13 # Establece la clave API de OpenAI desde una variable de entorno
14 openai.api_key = os.environ['OPENAI_API_KEY']
15
16 # Variable definida como una lista para almacenar objetos de mensaje
17 conversation_history = [initial_prompt]
18 |
19 # Define un modelo de datos con Pydantic que se utiliza para validar los datos de entrada en el endpoint /chat
20 class ChatRequest(BaseModel):
21     message: str # Espera una entrada de tipo string llamada "message".
22
23 # Función para enviar un mensaje al modelo de chat y registrar la conversación
24 def send_message_to_chat(message_text):
25     # Añade el mensaje del usuario al historial de la conversación
26     conversation_history.append({"role": "user", "content": message_text})
27     # Solicita una respuesta del modelo de OpenAI
28     response = openai.chat.completions.create(
29         model="gpt-3.5-turbo", # Modelo de OpenAI usado
30         messages=conversation_history,
31         tools=tools
32     )
33     print(f"Tokens used in the response: {response.usage.total_tokens}")
34     response_message = response.choices[0].message

```

- Configuración de la API de OpenAI para generar respuestas a través del modelo GPT-3.5-turbo.

```

35     # Registra el mensaje del usuario y la respuesta del asistente en el historial
36     conversation_history.append(response_message)
37     tool_calls = response_message.tool_calls
38     if tool_calls:
39         for tool_call in tool_calls:
40             function_name = tool_call.function.name
41             arguments_str = tool_call.function.arguments
42             print(f"Call function: {function_name} with arguments: {arguments_str}")
43             func_to_run = functions.get(function_name)
44             if func_to_run:
45                 function_response = func_to_run(arguments_str)
46                 print("Run function ... Result:", function_response)
47                 conversation_history.append({
48                     "tool_call_id": tool_call.id,
49                     "role": "tool",
50                     "name": function_name,
51                     "content": function_response,
52
53                 })
54     # Solicita una nueva respuesta tras ejecutar las herramientas
55     response = openai.chat.completions.create(
56         model="gpt-3.5-turbo",
57         messages=conversation_history,
58         tools=tools
59     )
60     conversation_history.append({"role": "assistant", "content": response.choices[0].message.content})
61     print(f"Tokens used in the response: {response.usage.total_tokens}")
62     return response.choices[0].message.content
63
64     else:
65         return response_message.content

```

- Funciones para enviar y registrar mensajes dentro del historial de conversaciones.

```

67     # Función asíncrona que genera una respuesta usando la función anterior
68     async def generate_response(user_input: str) -> str:
69         return send_message_to_chat(user_input)
70
71     # Formatea un diccionario a strings para mostrarlo de manera legible
72     def format_dict_to_strings(obj):
73         result = []
74         for key, value in obj.items():
75             result.append(f"{key}: {value}")
76         return '\n'.join(result)
77
78     # Analiza la respuesta del chat para determinar si se debe realizar una llamada a la base de datos
79     def parse_chat_response(response):
80         is_data_returned = False
81         return_obj = {}
82
83         print(f"parse_chat_response function called ... Body: {response}")
84         if ':::get_professors' in response:
85             is_data_returned = True
86             return_obj['get_professors'] = {
87                 "instructions": "This is the result of the querying database. The return value includes list of professors",
88                 "result": get_professors()
89             }
90         if is_data_returned:
91             return format_dict_to_strings(return_obj), is_data_returned
92         else:
93             return 'No additional data returned from the database'

```

- Funcionalidad para invocar operaciones de base de datos basadas en las solicitudes del usuario y manejar la respuesta dinámicamente.

7.3. *data_processors.py*

Este archivo está diseñado para manejar las operaciones de consulta de datos relacionadas con los usuarios de un centro educativo (profesores, estudiantes, grupos, etc.) utilizando la base de datos del sistema:

- Establece una sesión con la base de datos local utilizando SQLAlchemy, permitiendo la ejecución de consultas y transacciones.

```
python_apps > chatbot > 📁 data_processors.py > ...
1  import json
2  from datetime import date
3  from typing import Any
4  from sqlalchemy.orm import sessionmaker
5  from python_apps.models_sql import models
6  from python_apps.models_sql.database import SessionLocal
Documents/MyFirstLab/db_setup >rt BaseModel, field_validator
7
8
9  # Crea una sesión de base de datos local
10 session = SessionLocal()
11
```

- Funciones de Consulta de Datos:

get_professors: Recupera la lista de profesores de la base de datos. Opcionalmente, filtra los profesores según la asignatura que imparten.

get_students: Consulta la lista de todos los estudiantes, pudiendo filtrar por el grupo al que pertenecen.

get_groups: Obtiene una lista de todos los grupos educativos registrados en el sistema.

get_notas_of_alumno: Recupera las calificaciones de un estudiante específico basado en su nombre, devolviendo detalles como la asignatura, la calificación y la fecha de la evaluación.

get_schedule_for_date: Genera el horario de clases para una fecha específica, incluyendo detalles como la asignatura, grupo, tema, hora y día. Uso de SQLAlchemy para facilitar la consulta y manipulación de la base de datos.

```

27  # Recupera la lista de profesores desde la base de datos.
28 def get_professors(filter_obj: str = None):
29     result = session.query(models.Profesor).all()
30     search_result = []
31     # Si se obtienen resultados, procesar cada objeto para obtener los datos necesarios.
32     if result:
33         search_result = [{'name': obj.nombre, 'subject': obj.asignatura.nombre} for obj in result]
34     # Imprime los resultados de la función para propósitos de depuración.
35     print(f"Call get_professors func with parameter: {filter_obj} ... Result: {search_result}")
36     # Convierte la lista de diccionarios a un string JSON para su retorno.
37     return json.dumps(search_result)
38
39 # Recupera la lista de estudiantes desde la base de datos.
40 def get_students(filter_obj: str = None):
41     result = session.query(models.Alumno).all()
42     search_result = []
43     if result:
44         search_result = [{"name": obj.nombre, "group": obj.grupo.nombre} for obj in result]
45     print(f"Call get_students func with parameter: {filter_obj} ... Result: {search_result}")
46     return json.dumps(search_result)
47
48 #Recupera la lista de grupos desde la base de datos.
49 def get_groups(filter_obj: dict = {}):
50     result = session.query(models.Grupo).all()
51     search_result = []
52     if result:
53         search_result = [obj.nombre for obj in result]
54     print(f"Call get_groups func with parameter: {filter_obj} ... Result: {search_result}")
55     return json.dumps(search_result)

```

Todas las funciones convierten los resultados de las consultas a formatos JSON para facilitar su integración con otras partes del sistema, como la interfaz de usuario del chatbot.

Esta estructura modular facilita el mantenimiento y la escalabilidad del sistema, asegurando que el chatbot pueda responder eficazmente a las consultas de los usuarios y manejar los datos de manera eficiente.

7.7. ANEXO: Configuración de la Base de Datos y Generación de Datos en db_setup

En esta sección podemos observar la configuración y generación de datos para la base de datos de nuestro chatbot. Nos centraremos en los archivos `__init__.py`, `add_rows.py` y `horario_generator.py`, los cuales gestionan la inicialización y población de la base de datos.

La configuración y generación de datos es esencial para asegurar que la base de datos contenga la información necesaria para el funcionamiento del chatbot. Utilizaremos varios scripts para inicializar la base de datos y añadir datos relevantes.

7.1. `__init__.py`

Este archivo sirve como punto de entrada y configuración inicial para las funcionalidades relacionadas con la manipulación y consulta de datos en un sistema de gestión educativa. Se encarga de establecer la conexión con un servidor y define varias funciones para la interacción con este a través de peticiones HTTP.

- Establece la dirección IP y el puerto para el servidor que maneja las peticiones HTTP.

```
db_setup > 🐍 __init__.py > ...
1 import requests
2 import json
3
4 # Configuración de la dirección del servidor y el puerto para las solicitudes HTTP
5 service_host = '127.0.0.1'
6 service_port = 8095
7 |
```

- Funciones de Peticiones HTTP:

`post_request`: Envía peticiones POST al servidor y maneja las respuestas, incluyendo la gestión de errores y el cierre del programa en caso de fallo.

`get_request`: Realiza peticiones GET para recuperar datos del servidor, gestionando también los errores y cierres necesarios.

```

8   # Función para enviar peticiones POST a un servidor
9  ✓ def post_request(path, body):
10 ✓   try:
11     result = requests.post(f'http://{service_host}:{service_port}{path}', data=json.dumps(body))
12     # Verifica si la respuesta es satisfactoria
13     if result.status_code == 200:
14       return result.json()['id']
15     else:
16       # Imprime un mensaje de error si la solicitud falla
17       print(f"Request failed to {result.url}.\nStatus code: {result.status_code}. Reason: {result.reason}")
18       exit(0)
19   except requests.exceptions.HTTPError as exc:
20     print(exc)
21   return
22
23   # Función para enviar peticiones GET a un servidor
24 ✓ def get_request(path):
25 ✓   try:
26     result = requests.get(f'http://{service_host}:{service_port}{path}')
27     if result.status_code == 200:
28       return result.json()
29     else:
30       print(f"Request failed to {result.url}.\nStatus code: {result.status_code}. Reason: {result.reason}")
31       exit(0)
32   except requests.exceptions.HTTPError as exc:
33     print(exc)
34   return
35

```

- Recuperación de Identificadores:

asignatura_ids: Obtiene los identificadores de las asignaturas desde el servidor.

temarios_ids: Recupera los identificadores de los temarios disponibles.

grupos_ids: Obtiene los identificadores de los grupos educativos.

```

1   # Recupera los IDs de las asignaturas desde el servidor
2  ✓ def asignatura_ids():
3    objects = get_request('/asignaturas/search')
4    return {obj["nombre"]: obj["id"] for obj in objects}
5
6    # Recupera los IDs de los temarios desde el servidor
7  ✓ def temarios_ids():
8    objects = get_request('/temarios/search')
9    return {obj["tema"]: obj["id"] for obj in objects}
10
11
12  # Recupera los IDs de los grupos desde el servidor
13  ✓ def grupos_ids():
14    objects = get_request('/grupos/search')
15    return {obj["nombre"]: obj["id"] for obj in objects}
16

```

- Datos para Población de la Base de Datos:

```

51  # Datos para llenar tablas grupo y alumno de mi base de datos
52  grupos_y_alumnos = {
53      "ASIR": {
54          "alumnos": [
55              "Sofia Martínez López",
56              "Gonzalo Calvo del Torto",
57              "Lucia Gómez Sánchez",
58              "Alejandro Ramos Hernández",
59              "Valentina Fernández Ruiz",
60              "Carlos Navarro Díaz",
61              "Martina Alonso Moreno",
62              "Javier Muñoz Jiménez",
63              "Isabella Vega Pérez",
64              "Serhiy Holovasin",
65              "Adriana Ortiz Molina",
66              "Luis Guerra Aguilar"
67      ]

```

grupos_y_alumnos: Contiene información para poblar las tablas de grupos y alumnos en la base de datos.

```

103  # Datos para llenar tablas profesor, horario y temario de mi base de datos
104  horarios_y_temas = {
105      "Programación": {
106          "profesor": "Rafael Mendoza Castellano",
107          "temarios": [
108              "Introducción a la programación",
109              "Estructuras de datos",
110              "Algoritmos básicos",
111              "Programación orientada a objetos",
112              "Gestión de memoria",
113              "Recursividad",
114              "Principios SOLID",
115              "Patrones de diseño",
116              "Programación funcional",
117              "Desarrollo dirigido por pruebas",
118              "Refactorización de código",
119              "Integración continua",
120              "Despliegue de aplicaciones",
121              "Programación de sistemas",
122              "Desarrollo de APIs"
123      ],
124  },

```

horarios_y_temas: Almacena datos para llenar las tablas de profesores, horarios y temarios.

- Ejecución de Configuración de la Base de Datos:

Al ejecutar el script directamente, se invocan funciones de otros módulos para agregar datos a la base de datos, generar horarios y notas para los alumnos.

7.2. *add_rows.py*:

Este archivo contiene funciones específicas para poblar la base de datos de un sistema de gestión educativa con datos iniciales. Las funciones están diseñadas para interactuar con un servidor a través de peticiones HTTP, utilizando la librería requests para agregar grupos, alumnos, profesores, asignaturas, temarios y horarios.

- Función *add_grupos_y_alumnos*:

```
db_setup > 🐍 add_rows.py > ⚙️ add_grupos_y_alumnos
1   import json
2   from db_setup import post_request, grupos_y_alumnos, horarios_y_temas, horarios
3
4   # Función para agregar grupos y alumnos al BD
5   def add_grupos_y_alumnos():
6       # Itera sobre cada grupo y su lista de alumnos
7       for group, alumnos_list in grupos_y_alumnos.items():
8           group_id = post_request('/grupos/create', {"nombre": group})
9           print(f"Grupo '{group}' created. Id: '{group_id}'")
10          if group_id:
11              # Si el grupo se creó con éxito, procede a agregar alumnos a ese grupo
12              for alumno in alumnos_list['alumnos']:
13                  # Crea un alumno en la base de datos y obtiene el ID del alumno
14                  alumno_id = post_request('/alumnos/create', {
15                      "nombre": alumno,
16                      "grupo_id": group_id
17                  })
18                  if alumno_id:
19                      print(f"Alumno '{alumno}' created. Id: '{alumno_id}'")
20
```

Agrega grupos y alumnos a la base de datos. Para cada grupo, se envía una petición POST para crear el grupo y obtener un ID. Luego, para cada alumno del grupo, se envía otra petición POST para agregar el alumno con referencia al ID del grupo.

- Función *add_profesores_asignaturas_temas*:

Agrega profesores, asignaturas y temarios relacionados a la base de datos. Al crear la asignatura, se agrega un profesor asignado a esa asignatura y se agregan múltiples temarios asociados.

```

21  # Función para agregar profesores, asignaturas y temarios al BD
22  def add_profesors_asignaturas_temas():
23      for asignatura, asignatura_items in horarios_y_temas.items():
24          profesor = asignatura_items['profesor']
25          temarios = asignatura_items['temarios']
26          # Crea una asignatura en la base de datos y obtiene el ID de la asignatura
27          asignatura_id = post_request(
28              '/asignaturas/create',
29              {"nombre": asignatura}
30          )
31          print(f"Asignatura: '{asignatura} (id: {asignatura_id})' successfully added to DB")
32
33          # Crea un profesor asignado a esa asignatura en la base de datos
34          profesor_id = post_request('/profesors/create', {
35              "nombre": profesor,
36              "asignatura_id": asignatura_id
37          })
38          print(f"Profesor: '{profesor} (id: {profesor_id})' successfully added for '{asignatura}'")
39
40          # Itera sobre cada tema relacionado con la asignatura
41          for tema in temarios:
42              # Crea un tema en la base de datos y obtiene el ID del tema
43              tema_id = post_request('/temarios/create', {
44                  "tema": tema,
45                  "asignatura_id": asignatura_id
46              })
47              print(f"Tema: '{tema} (id: {tema_id})' successfully added for '{asignatura}'")
48

```

- Función *add hora ids*:

Agrega horarios predefinidos al sistema. Envía peticiones POST para cada horario, creando registros de horarios en la base de datos..

```

49  # Función para agregar horarios al sistema
50  def add_hora_ids():
51      # Itera sobre cada horario predefinido
52      for hora in horarios:
53          # Crea un horario en la base de datos
54          post_request('/horas/create', {"hora": hora})
55

```

7.3. *horario_generator.py*:

Este archivo es responsable de la generación y manejo de horarios para un centro educativo, utilizando estructuras de datos complejas y operaciones sobre fechas para distribuir asignaturas, temas y horarios entre grupos de estudiantes. También maneja la creación de notas aleatorias para los alumnos.

- *Manejo de Fechas y Festivos:*

```

2 import datetime
3 import json
4 import random
5 import itertools
6 from collections import OrderedDict
7 from db_setup import post_request, get_request, grupos_y_alumnos, horarios_y_temas, asignatura_ids, grupos_ids, temarios_ids
8
9 # Conjuntos de grupos y horarios obtenidos de las estructuras de datos
10 groups_set = set(grupos_y_alumnos.keys())
11 horarios_set = set(horarios_y_temas.keys())
12
13
14 # Lista de días festivos públicos en la Comunidad de Madrid, España, para el año 2024, para generar horario
15 public_holidays_2024_madrid = [
16     datetime.date(2024, 1, 1), # Año Nuevo
17     datetime.date(2024, 1, 6), # Día de Reyes o Epifanía del Señor
18     datetime.date(2024, 3, 29), # Viernes Santo
19     datetime.date(2024, 5, 1), # Día del Trabajador
20     datetime.date(2024, 5, 2), # Día de la Comunidad de Madrid
21     datetime.date(2024, 8, 15), # Asunción de la Virgen
22     datetime.date(2024, 10, 12), # Fiesta Nacional de España
23     datetime.date(2024, 11, 1), # Día de Todos los Santos
24     datetime.date(2024, 12, 6), # Día de la Constitución Española
25     datetime.date(2024, 12, 8), # Inmaculada Concepción
26     datetime.date(2024, 12, 25) # Día de Navidad
27 ]

```

```

def is_weekend_or_holiday(date):
    """ Compruebe si la fecha es fin de semana o festivo. """
    return date.weekday() >= 5 or date in public_holidays_2024_madrid

def find_start_date(year):
    """ Busca el primer día laborable de septiembre del año en cuestión. """
    date = datetime.date(year, 9, 1)
    while is_weekend_or_holiday(date):
        date += datetime.timedelta(days=1)
    return date

```

Identifica los días festivos y fines de semana para evitar programar clases en estas fechas. Busca el primer día laborable de septiembre de un año dado para iniciar el calendario académico.

- Generación de Horarios:

```

def refined_round_robin_topic_generator(shift_index):
    """
    Generador para iterar sobre cada curso de manera rotativa y generar temas de forma cíclica
    hasta agotar todos los temas de todos los cursos.

    shifted_dict = shift_keys(shift_index)
    # Crea una lista de tuplas (course_name, topic_iterator), incluyendo solo listas de temas no vacías
    iterators = [(course, iter(details.get("temarios", []))) for course, details in shifted_dict.items() if "temarios" in details and details["temarios"]]

    # Cicla a través de los iteradores de cursos de forma indefinida, parando cuando todos se hayan agotado
    active_iterators = len(iterators)
    if active_iterators == 0:
        return

    # Utiliza itertools.cycle para gestionar la rotación de los iteradores
    cycle_iterators = itertools.cycle(iterators)
    while active_iterators > 0:
        course, topic_iter = next(cycle_iterators)
        try:
            # Devuelve el siguiente tema del iterador del curso actual
            yield (course, next(topic_iter))
        except StopIteration:
            # Este iterador se ha agotado, remuévelo del ciclo para evitar más intentos de obtención de temas
            active_iterators -= 1
            iterators.remove((course, topic_iter))
            cycle_iterators = itertools.cycle(iterators) # Re-crea el ciclo con los iteradores restantes

```

Utiliza un enfoque de round robin refinado para asignar temas de manera equitativa entre los cursos y grupos.

Genera un horario completo asignando asignaturas y temas específicos para cada grupo de estudiantes, ajustando las asignaciones según la disponibilidad de los slots.

```
def generate_schedule(start_date, groups, subjects):
    """ Genera un horario escolar para un año a partir de una fecha de inicio, agrupando por grupos y asignaturas. """
    schedule = {}
    # Inicializa índices de tema para cada asignatura
    topic_index = {subject: 0 for subject in subjects}
```

```
160 def create_horario():
161     """ Crea el horario completo asignando asignaturas y temas a cada grupo de estudiantes en slots disponibles."""
162     # Obtiene IDs de las asignaturas, temas, grupos desde un servicio externo
163     asignatura_name_to_id = asignatura_ids()
164     tema_name_to_id = temarios_ids()
165     grupo_name_to_id = grupos_ids()
166     total_number_of_lessons_needed = calculate_total_count_of_lessons()
167     group_iterators = create_generators_per_group()
168     lesson_number = 0
169     slots = generate_lesson_available_slots()
170     while lesson_number < total_number_of_lessons_needed:
171         for group in grupos_y_alumnos.keys():
172             try:
173                 asignatura, tema = next(group_iterators[group])
174                 print(f"{lesson_number}. {slots[lesson_number]} {group} - {asignatura.upper()}: '{tema}'")
175                 horario_obj = {
176                     "asignatura_id": asignatura_name_to_id[asignatura],
177                     "grupo_id": grupo_name_to_id[group],
178                     "dia": slots[lesson_number][0],
179                     "hora_id": slots[lesson_number][1],
180                     "tema_id": tema_name_to_id[tema]
181                 }
182                 print(horario_obj)
183                 post_request("/horarios/create", horario_obj)
184             except StopIteration:
185                 print("No more items in the iterator.")
186             lesson_number += 1
187
```

- Creación de Notas:

```
189 def create_notas():
190     """ Crea notas aleatorias para alumnos en los horarios creados."""
191     horarios = get_request("/horarios/search")
192     alumnos = get_request('/alumnos/search')
193     alumnos_per_group = {}
194     counter = 0
195     for alumno in alumnos:
196         if alumno['grupo_id'] not in alumnos_per_group:
197             alumnos_per_group[alumno["grupo_id"]] = []
198             alumnos_per_group[alumno["grupo_id"]].append(alumno["id"]) # Agrupa alumnos por grupo
199     for horario in horarios:
200         grupo_id = horario["grupo_id"]
201         horario_id = horario["id"]
202         random_alumnos = random.sample(alumnos_per_group[grupo_id], int(len(alumnos_per_group[grupo_id])/3))
203         for alumno_id in random_alumnos:
204             print(f"{counter}. Add Nota for alumno_id: '{alumno_id}'. horario_id: '{horario_id}'")
205             post_request("/notas/create", {
206                 "alumno_id": alumno_id,
207                 "horario_id": horario_id,
208                 "nota": random.randint(4, 10) # Asigna una nota aleatoria entre 4 y 10
209             })
210             counter += 1
```

Asigna notas aleatorias a los alumnos en los horarios creados, utilizando una selección aleatoria de alumnos y asignándoles calificaciones entre 4 y 10.

- Rotación de Temas:

Implementa un generador que itera sobre cada curso de manera rotativa para asignar temas de forma cíclica.

- Funciones de Soporte:

Calcula el total de lecciones necesarias multiplicando el número de temas por el número de grupos. Crea generadores de temas por grupo usando un desplazamiento inicial para cada grupo, facilitando la distribución diversa de los temas.

Genera y asigna slots disponibles para lecciones basándose en la disponibilidad y necesidades totales de lecciones.

La configuración de la base de datos y la generación de datos en el proyecto del chatbot educativo asegura que la base de datos contenga la información necesaria para el funcionamiento del sistema.

7.8. ANEXO: Adquisición de dominio

Hostalia.com es un proveedor de servicios de hosting e Internet que ofrece una variedad de soluciones para empresas y particulares, incluyendo registro de dominios, alojamiento web, servidores virtuales y dedicados, y otros servicios relacionados. Esta empresa es conocida por su accesibilidad y una amplia gama de opciones que se adaptan a las necesidades de diferentes usuarios, desde pequeños proyectos personales hasta grandes infraestructuras empresariales.

En mi proyecto, utilizo Hostalia para registrar y gestionar el dominio *awsproject.es* debido a las restricciones presupuestarias y de permisos en el entorno de laboratorio de AWS.

The screenshot shows the 'Gestión de dominios y SSL' section of the Hostalia website. It displays a table with the following information:

| DOMINIOS | CERTIFICADOS SSL | AGENDA DE CONTACTOS | REGISTRO DE ACCIONES | |
|---|--|--|---|--|
| Dominio awsproject.es Aditional | Estate Registrado hasta 07/04/2025 | Renovación Automática | Usuario serhiyhol | Usado en Plan Dominio 40905696 |
| SSL | | | | |
| REGISTRO | SERVICIOS | | | |
| Estado: Registrado Nombre: awsproject.es Expiración: 07/04/2025 Creación: 07/04/2024 Renovación: Automática | Tipo: Aditional En: Plan Dominio 40905696 Pago: PayPal Usuario: serhiyhol | Datos del dominio: Titular Nombre: E-mail: Dirección: Teléfono: | Servidores DNS: DNS 1: ns10.servicio-online.net DNS 2: ns11.servicio-online.net DNS 3: ns12.servicio-online.net | |
| RENOVAR CAMBIAR USUARIO | | CAMBIAR DNS | | |

Hostalia ofrece la ventaja de ser una opción coste-efectiva para registrar dominios (0,0 para los nuevos usuarios y dominios *.es) , además de proporcionar servicios adicionales como DNS y certificados SSL, que son esenciales para la correcta configuración y seguridad de nuestro sitio web. El hecho de que Hostalia ofrezca certificados SSL de DigiCert con la compra del dominio también simplifica el proceso de asegurar nuestra web, lo cual es importante para mantener la confianza de los usuarios y proteger la transmisión de datos.

The screenshot shows the 'Secure Site Starter' certificate details for the domain awsproject.es. The table includes the following information:

| | | | | | |
|--|---|---|------------|-----------|-----------------------|
| Secure Site Starter Incluido | awsproject.es | Renovado hasta 06/04/2025 | Automática | serhiyhol | Plan Dominio 40905697 |
| DETALLES | | | | | |
| Certificado: Secure Site Starter | Estado: Renovado | Datos del certificado: Titular | | | |
| Dominio: awsproject.es Validación: Dominio (DV) Protección: Dominio Firma CA: DigiCert Garantía: 0 USD | Emitido el: 07/04/2024 Válido hasta: 06/04/2025 Precio anual: 0,00€ Usado en: Plan Dominio Renovación: Automática | Nombre: Serhiy Holovasin Apellidos: Email: Empresa: País: | | | |
| VER DATOS | | | | | |

Al trabajar con AWS y Hostalia juntos, podemos aprovechar la robusta infraestructura y las capacidades de entrega de contenido de AWS, mientras gestionamos nuestras necesidades de dominio y SSL a través de Hostalia.

Como resultado, se adquirió el dominio *awsproject.es* a través de Hostalia, aprovechando sus precios competitivos y la facilidad de gestión de dominios.

Esta estrategia resulta ser adecuada para un entorno de laboratorio con limitaciones de presupuesto y permisos, asegurando que el proyecto se mantenga dentro de los límites financieros sin comprometer la funcionalidad o la seguridad.

7.9. ANEXO: Configuración de Frontend en AWS con S3 y CloudFront

El objetivo de este anexo es detallar el proceso de despliegue y configuración del frontend de este proyecto en AWS, utilizando Amazon S3 para alojamiento de archivos estáticos y Amazon CloudFront como CDN para distribuir el contenido de manera eficiente y segura.

Configuración de Amazon S3

1. Creación de Bucket:

Se crea un bucket en Amazon S3 con el nombre del instituto, web-ies-infanta-elena, para mantener una relación clara con el dominio utilizado. Se configura para alojar un sitio web estático, especificando index.html como documento de índice y el archivo de error correspondiente si es necesario.

The screenshot shows the AWS S3 Buckets page. A green banner at the top indicates that the bucket 'web-ies-infanta-elena' was created successfully. Below the banner, the page displays the bucket's details: name, region (us-east-1), and creation date (May 18, 2024). The 'Buckets de uso general' tab is selected, showing one item: 'web-ies-infanta-elena'. There are buttons for actions like 'Copiar ARN', 'Vaciar', 'Eliminar', and 'Crear bucket'.

2. Carga de Archivos:

Se suben los archivos del frontend (HTML, CSS, JavaScript, imágenes) al bucket.

The screenshot shows the AWS S3 Objects page for the 'web-ies-infanta-elena' bucket. It lists four objects: 'index.html', 'script.js', 'style.css', and 'img/'. The 'index.html' file is identified as the 'Documento de índice'. The 'script.js' and 'style.css' files are identified as 'Archivos de error'. The 'img/' folder is identified as ' Carpeta'. There are buttons for actions like 'Copiar URI de S3', 'Copiar URL', 'Descargar', 'Abrir', 'Eliminar', 'Acciones', 'Crear carpeta', and 'Cargar'.

Se configuran los permisos de los archivos para asegurar que son accesibles públicamente, permitiendo su visualización en internet.

Editar la política del bucket [Información](#) [Información](#)

Política de bucket

La política del bucket, escrita en JSON, proporciona acceso a los objetos almacenados en el bucket. La

ARN del bucket

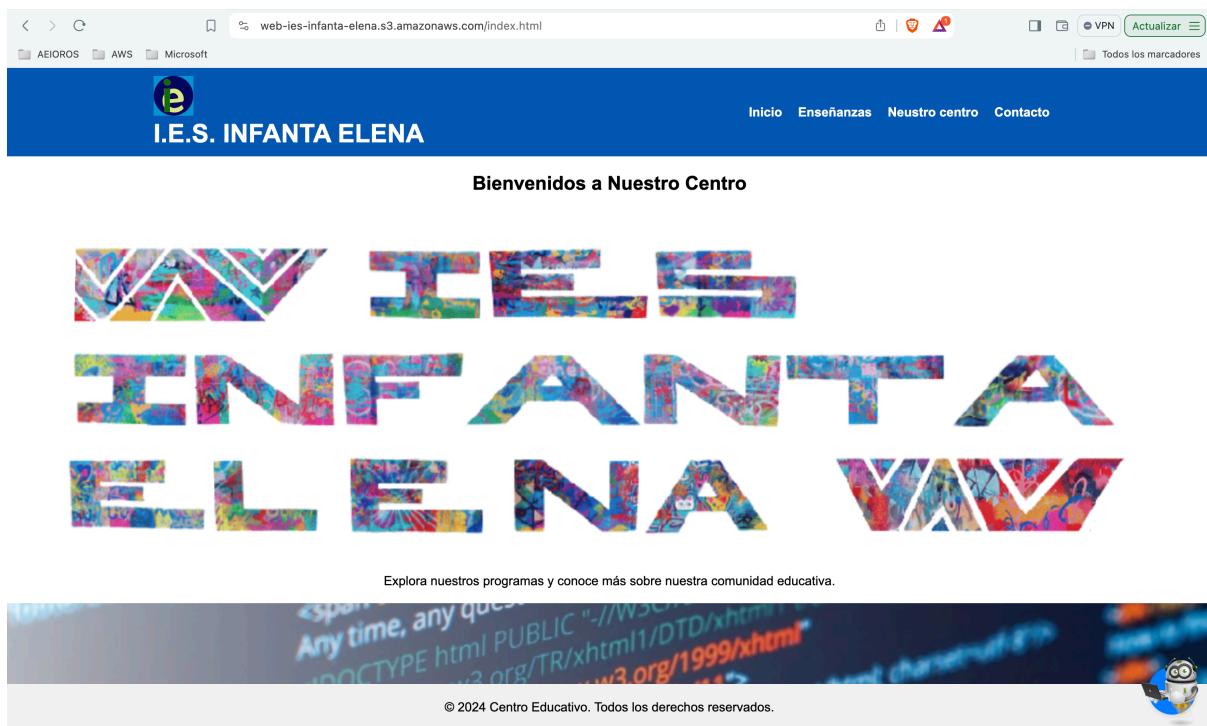
arn:aws:s3:::web-ies-infanta-elena

Política

```

1 ▼ {
2   "Version": "2012-10-17",
3 ▼   "Statement": [
4     {"Sid": "PublicReadGetObject",
5      "Effect": "Allow",
6      "Principal": "*",
7      "Action": ["s3:GetObject"],
8      "Resource": ["arn:aws:s3:::web-ies-infanta-elena/*"]
9    ]
10   ]
11 }
12 }
13 }
```

Entrando al URL del objeto, comprobamos que el sitio web es accesible públicamente.



Configuración de Amazon CloudFront

1. Creación de la distribución:

Se crea una distribución de CloudFront seleccionando el bucket de S3 como origen. Comprobamos que con el dominio que nos proporciona Amazon CloudFront llegamos a nuestro sitio web:

Configuración de DNS en Hostalia

1. Registro A:

Dado que Hostalia asigna una dirección IP estática, se configura un registro A en el panel de DNS de Hostalia para que apunte a esta dirección IP y un registro CNAME:

| LISTA ENTRADAS DNS | | | |
|--------------------|--------------------|-------|---|
| | Entrada | Tipo | Valor |
| A | awsproject.es | A | 217.116.5.231 |
| NS | awsproject.es | NS | ns10.servicio-online.net. |
| NS | awsproject.es | NS | ns11.servicio-online.net. |
| NS | awsproject.es | NS | ns12.servicio-online.net. |
| MX 10 | awsproject.es | MX 10 | mx.awsproject.es. |
| TXT | awsproject.es | TXT | v=spf1 redirect=spf.dominioabsoluto.net |
| A | imap.awsproject.es | A | 217.116.0.237 |
| A | mx.awsproject.es | A | 217.116.0.227 |
| A | pop3.awsproject.es | A | 217.116.0.237 |

| LISTA ENTRADAS DNS | | | |
|--------------------|-----------------------|-------|----------------|
| | Entrada | Tipo | Valor |
| A | webmail.awsproject.es | A | 217.116.0.155 |
| CNAME | www.awsproject.es | CNAME | awsproject.es. |

2. Redireccionamiento:

Se configura un redireccionamiento en Hostalia para que todas las solicitudes a awsproject.es se redirijan a la URL de CloudFront proporcionada, d2qe9ij5atzw7k.cloudfront.net asegurando que el tráfico sea manejado eficientemente por el CDN.

Destino Web

Está gestionando el destino web de: [awsproject.es](#)

Seleccione nuevo destino

- Página Web
- Página de Cortesía
- Redirección Web (destino actual)

Cambiar tipo de Redirección*

Mantiene URL (marcos)

Redirigir a URL o IP *

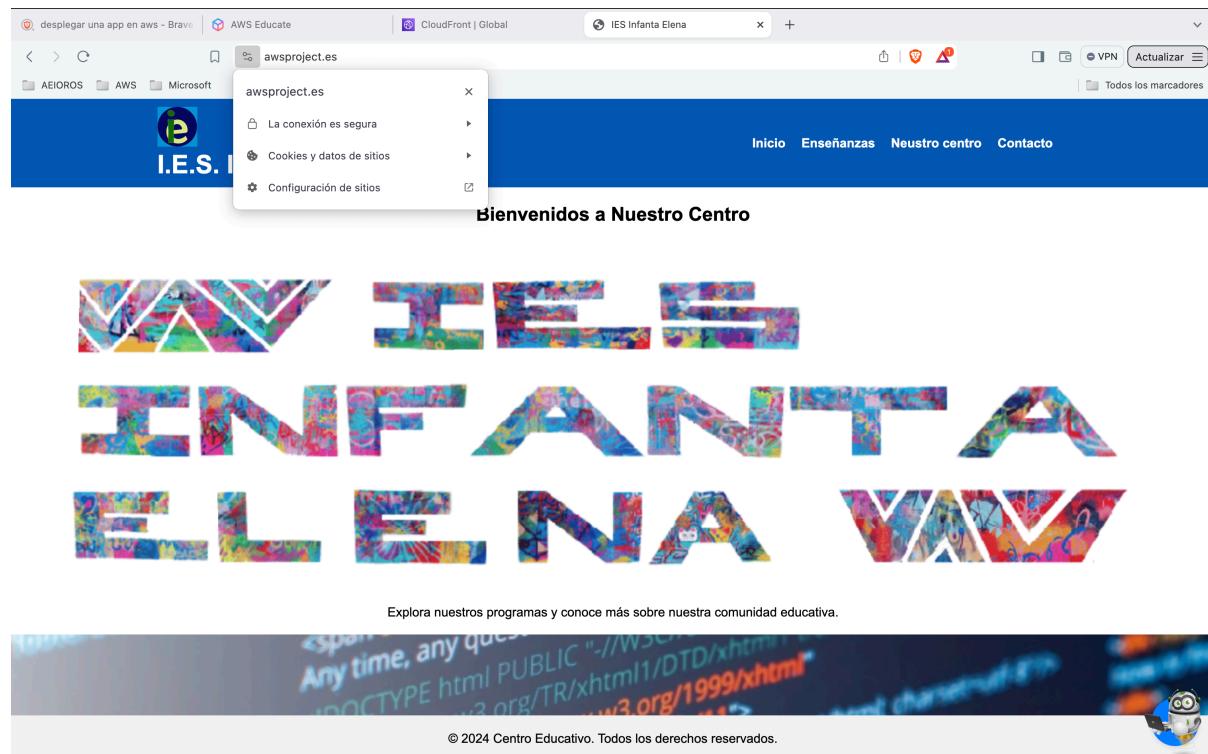
Opcional

Título de la página

IES Infanta Elena

Pruebas y Verificaciones:

Se realizan pruebas para asegurar que el sitio web es accesible tanto por HTTP como por HTTPS y que el certificado SSL está funcionando correctamente.





Bienvenidos a Nuestro Centro



Se utiliza herramientas como *dig* para verificar la correcta configuración de los registros DNS.

```
serhiy ~ serhiy@MacBook-Pro-de-Serhiy ~ ~ -zsh - 80x24
Last login: Thu May 16 20:19:44 on ttys007
21:16:43
> dig awsproject.es +noall +answer

; <>> DiG 9.10.6 <>> awsproject.es +noall +answer
;; global options: +cmd
awsproject.es.        43200    IN      A       217.116.5.231
21:16:47
>
```

Este anexo ofrece una visión comprensiva y técnica de cómo se configuró la infraestructura para servir el frontend del proyecto en AWS, asegurando la máxima eficiencia y escalabilidad.

7.10.ANEXO: Construcción de Imagen Docker para el Chatbot

1. Preparación del Archivo Dockerfile:

El primer paso es crear un Dockerfile, que es un archivo de texto que contiene todas las instrucciones necesarias para construir la imagen Docker. Este archivo define el sistema operativo base, las dependencias, los archivos de nuestra aplicación y los comandos de ejecución.

```

python_apps > 📄 Dockerfile > ...
1  # Use an official Python runtime as the base image
2  FROM tiangolo/unicorn-gunicorn-fastapi:python3.11
3
4  # Set environment variables (optional)
5  ENV APP_HOME=/app
6  ENV PORT=8095
7
8  # Set the working directory in the container
9  WORKDIR $APP_HOME
10
11 # Add $APP_HOME to PYTHONPATH
12 ENV PYTHONPATH="${APP_HOME}"
13
14 # Copy the current directory contents into the container at $APP_HOME
15 COPY . $APP_HOME/python_apps
16
17 # Install Uvicorn (optional, as it should be included in the base image)
18 RUN pip install uvicorn
19
20 # Install any needed dependencies specified in requirements.txt
21 RUN pip install --no-cache-dir -r python_apps/requirements.txt
22
23 # Set environment variable
24 ENV OPENAI_API_KEY=${OPENAI_API_KEY}
25
26 # Make port 8080 available to the world outside this container
27 EXPOSE $PORT
28
29 # Command to run the FastAPI application
30 CMD ["uvicorn", "python_apps.main:app", "--host", "0.0.0.0", "--port", "8095"]

```

A continuación se detalla cada componente del Dockerfile y su función en la creación y configuración del contenedor.

- *Imagen Base:*

FROM tiangolo/unicorn-gunicorn-fastapi:python3.11: Utiliza una imagen base oficial de Python con Uvicorn y Gunicorn optimizados para FastAPI. Esta imagen está preconfigurada para ejecutar aplicaciones FastAPI de manera eficiente y segura.

Variables de Entorno:

ENV APP_HOME=/app: Establece una variable de entorno que define el directorio principal de la aplicación dentro del contenedor.

ENV PORT=8095: Define el puerto en el que se expone la aplicación FastAPI, configurado para escuchar en el puerto 8095.

Directorio de Trabajo:

WORKDIR \$APP_HOME: Establece el directorio de trabajo en el contenedor donde se ejecutarán los comandos subsiguientes. En este caso, todos los comandos se ejecutarán en /app.

PATH de Python:

ENV PYTHONPATH="\${APP_HOME}": Añade el directorio de la aplicación al PYTHONPATH, lo que permite que Python reconozca automáticamente los módulos dentro de este directorio.

Copiar Archivos:

COPY . \$APP_HOME/python_apps: Copia los contenidos del directorio actual (donde se encuentra el Dockerfile) en el directorio /app/python_apps dentro del contenedor. Esto incluye todos los archivos necesarios para ejecutar la aplicación.

Instalación de Dependencias:

RUN pip install unicorn: Instala Unicorn utilizando pip. Aunque Unicorn ya viene preinstalado en la imagen base, esta línea asegura que se utilice la última versión o se reinstale si fuese necesario.

RUN pip install --no-cache-dir -r python_apps/requirements.txt: Instala las dependencias necesarias para la aplicación, especificadas en requirements.txt, sin guardar caché para minimizar el tamaño del contenedor.

Variable de Entorno de OpenAI:

ENV OPENAI_API_KEY=\${OPENAI_API_KEY}: Configura una variable de entorno para la clave API de OpenAI, que será utilizada por la aplicación para interactuar con los servicios de OpenAI.

Exponer Puerto:

EXPOSE \$PORT: Expone el puerto configurado (8095) al exterior del contenedor, permitiendo el acceso a la aplicación FastAPI.

Comando de Ejecución:

CMD ["uvicorn", "python_apps.main:app", "--host", "0.0.0.0", "--port", "8095"]: Define el comando que se ejecutará automáticamente cuando se inicie el contenedor. Este comando inicia el servidor Uvicorn con la aplicación FastAPI, disponible en todos los interfaces de red del contenedor en el puerto 8095.

2. Construcción de la Imagen:

Una vez que el Dockerfile está preparado, podemos construir la imagen Docker ejecutando el siguiente comando en la terminal, desde el directorio donde se encuentra el Dockerfile:

```
~/Documents/MyFirstLab main
> cd python_apps

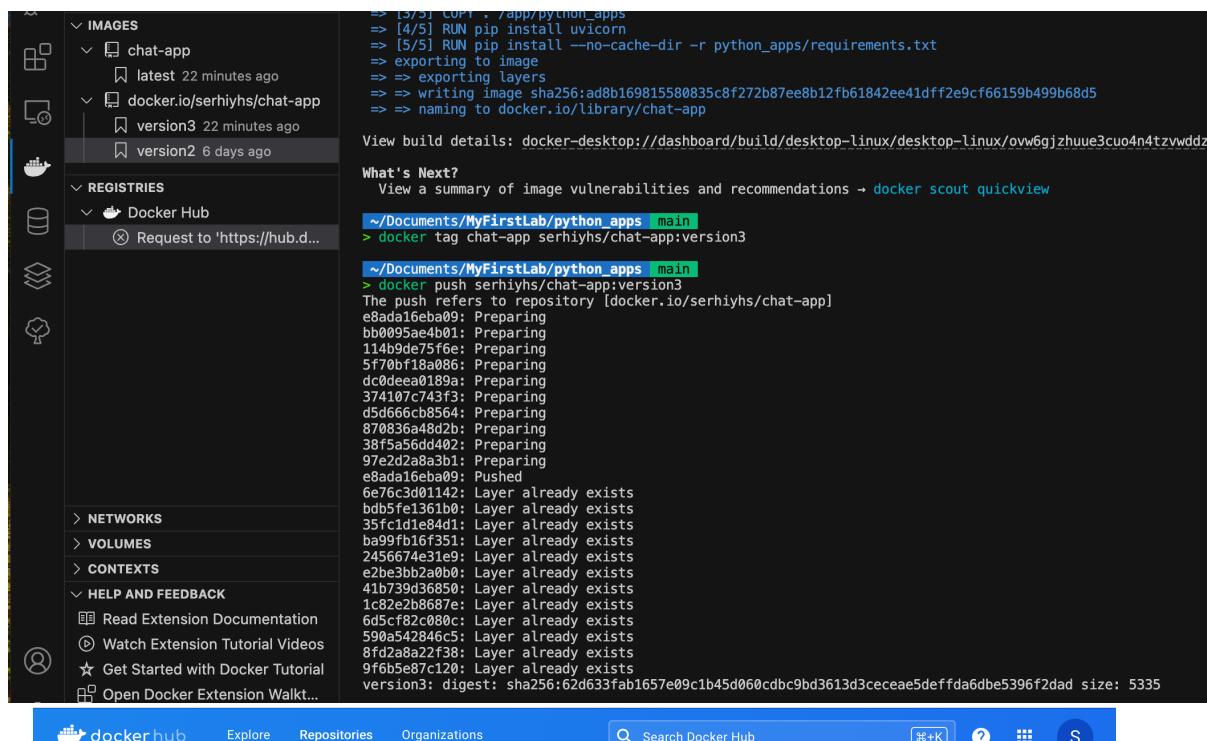
~/Documents/MyFirstLab/python_apps main
> docker build -t chat-app .
[+] Building 6.6s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 924B
=> [internal] load metadata for docker.io/tiangolo/uvicorn-gunicorn-fastapi:python3.11
=> [auth] tiangolo/uvicorn-gunicorn-fastapi:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/tiangolo/uvicorn-gunicorn-fastapi:python3.11@sha256:5b13f5130f1e70751e124305097f06438c
=> [internal] load build context
=> => transferring context: 126.75kB
=> CACHED [2/5] WORKDIR /app
=> [3/5] COPY . /app/python_apps
=> [4/5] RUN pip install uvicorn
=> [5/5] RUN pip install --no-cache-dir -r python_apps/requirements.txt
=> exporting to image
=> => exporting layers
=> => writing image sha256:ad8b169815580835c8f272b87ee8b12fb61842ee41dff2e9cf66159b499b68d5
=> => naming to docker.io/library/chat-app

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/ovw6gjzhuue3cuo4n4tzvwddz

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
```

3. Publicación de la Imagen en Docker Hub:

Docker utiliza etiquetas para organizar versiones de imágenes. Antes de subir nuestra imagen, debemos etiquetarla con nuestro nombre de usuario de Docker Hub y el nombre que deseamos darle, en mi caso utilice versiones. Una vez etiquetada, podemos subir nuestra imagen a Docker Hub:



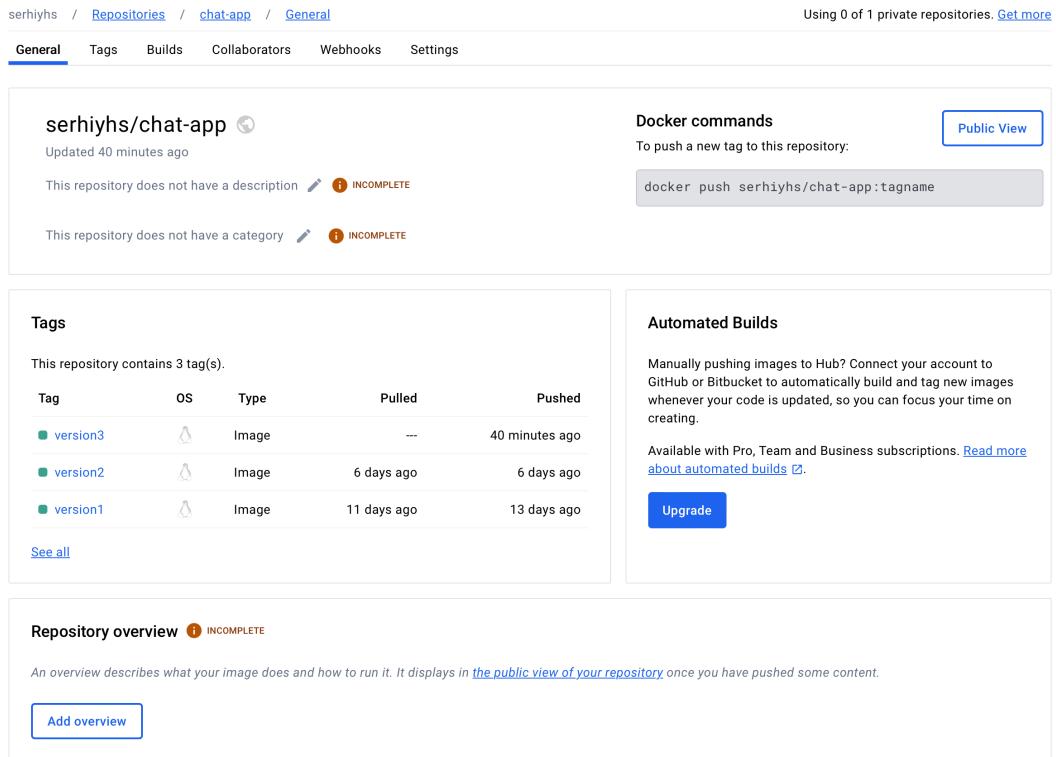
```

--> [3/3] COPY . /app/python_apps
--> [4/5] RUN pip install uvicorn
--> [5/5] RUN pip install --no-cache-dir -r python_apps/requirements.txt
--> exporting to image
--> >> exporting layers
--> >> writing image sha256:ad8b169815580835c8f272b87ee8b12fb61842ee41dff2e9cf66159b499b68d5
--> >> naming to docker.io/library/chat-app

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/ovw6gjzhuue3cuo4n4tzvwddz

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
~/Documents/MyFirstLab/python_apps main
> docker tag chat-app serhiyhs/chat-app:version3
~/Documents/MyFirstLab/python_apps main
> docker push serhiyhs/chat-app:version3
The push refers to repository [docker.io/serhiyhs/chat-app]
e8ada16eba09: Preparing
bb0095ae4b01: Preparing
114b9d7e75f6e: Preparing
5f70bf18a086: Preparing
dc0deea0189a: Preparing
374107c743f3: Preparing
d5d666cb8564: Preparing
870836a48d2b: Preparing
38f5a56dd402: Preparing
97e2d2a8a3b1: Preparing
e8ada16eba09: Pushed
6e76c3d01142: Layer already exists
bdb5fe1361b0: Layer already exists
35fc1d1e84d1: Layer already exists
ba99fb16f351: Layer already exists
2456674e31e9: Layer already exists
e2be3b2a0b0: Layer already exists
41b739d36850: Layer already exists
1c82e2b8687e: Layer already exists
6d5cf82c080c: Layer already exists
590a542846c5: Layer already exists
8fd2a8a22f38: Layer already exists
9f6b5e87c120: Layer already exists
version3: digest: sha256:62d633fab1657e09c1b45d060cdb9bd3613d3ceceae5defda6dbe5396f2dad size: 5335

```

| Tag | OS | Type | Pulled | Pushed |
|----------|----|-------|-------------|----------------|
| version3 | | Image | --- | 40 minutes ago |
| version2 | | Image | 6 days ago | 6 days ago |
| version1 | | Image | 11 days ago | 13 days ago |

7.11.ANEXO: Uso de la API de GPT-3.5 Turbo de OpenAI

Para el desarrollo de nuestro chatbot educativo, hemos integrado la versión GPT-3.5 Turbo de la API de OpenAI. Este modelo de inteligencia artificial, conocido por su potente rendimiento en procesamiento de lenguaje natural, es fundamental para permitir interacciones fluidas y inteligentes dentro del chatbot.

La GPT-3.5 Turbo es una versión mejorada del modelo de lenguaje GPT-3, diseñada para ofrecer respuestas más rápidas y precisas. Utiliza redes neuronales avanzadas que han sido entrenadas con vastos conjuntos de datos para entender y generar lenguaje humano de manera efectiva.

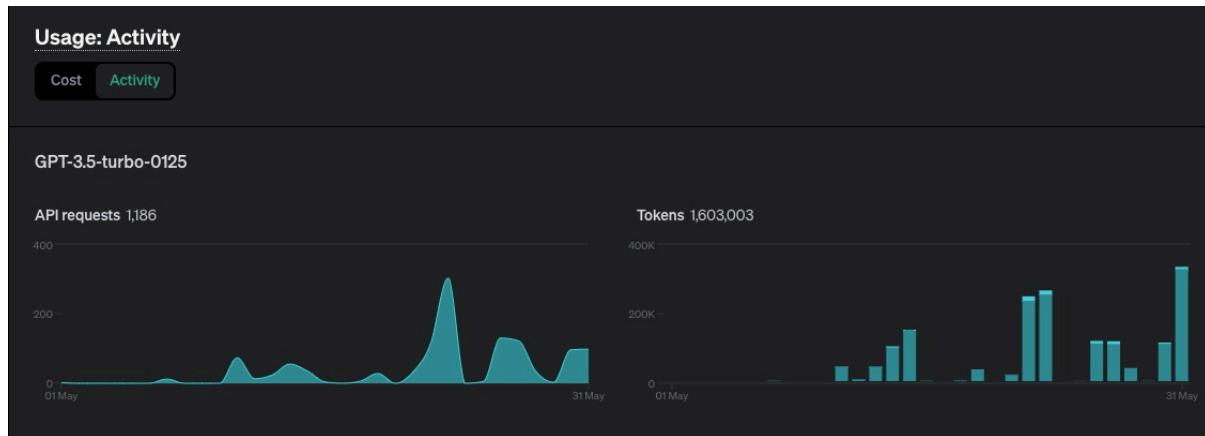
La incorporación de esta tecnología en nuestro chatbot permite:

- *Interacciones naturales*: Entender y responder a las preguntas de los usuarios de manera más natural y contextual.
- *Respuestas rápidas*: Procesar y responder a las consultas con menor latencia, lo cual es vital para mantener una experiencia de usuario fluida.
- *Capacidad de aprendizaje*: Mejorar continuamente a partir de nuevas interacciones, lo que aumenta la utilidad del chatbot con el tiempo.

El uso de GPT-3.5 Turbo implica costos basados en el número de tokens procesados (palabras o partes de palabras). OpenAI ofrece un esquema de precios que calcula el costo por cada mil tokens procesados. Aquí demuestro lo que he gastado en pruebas en mes de mayo:



Dado que la versión Turbo es más rápida y potente, puede llevar a un costo más alto comparado con las versiones anteriores. Los precios están sujetos a cambios y es recomendable consultar la página oficial de OpenAI para obtener información actualizada y detallada sobre tarifas.



Aunque GPT-3.5 Turbo es extremadamente poderoso, presenta algunas limitaciones que deben considerarse:

- *Costo por uso:* Para proyectos con un alto volumen de interacciones, el costo puede escalar rápidamente.
- *Limitaciones de Token:* El modelo GPT-3.5 Turbo de OpenAI impone un límite máximo de 16,385 tokens por solicitud. Este límite se refiere al número total de tokens que se pueden incluir en una sola entrada de texto (input) y salida de texto (output). Un token generalmente equivale a aproximadamente 4 caracteres de texto en inglés, lo que significa que una solicitud puede contener hasta aproximadamente 65,540 caracteres de texto, dependiendo de la complejidad y estructura del lenguaje utilizado.

GPT-3.5 Turbo

GPT-3.5 Turbo models can understand and generate natural language or code and have been optimized for chat using the [Chat Completions API](#) but work well for non-chat tasks as well.

| MODEL | DESCRIPTION | CONTEXT WINDOW | TRAINING DATA |
|--------------------|---|----------------|----------------|
| gpt-3.5-turbo-0125 | <p>New Updated GPT 3.5 Turbo</p> <p>The latest GPT-3.5 Turbo model with higher accuracy at responding in requested formats and a fix for a bug which caused a text encoding issue for non-English language function calls. Returns a maximum of 4,096 output tokens. Learn more.</p> | 16,385 tokens | Up to Sep 2021 |
| gpt-3.5-turbo | Currently points to gpt-3.5-turbo-0125. | 16,385 tokens | Up to Sep 2021 |
| gpt-3.5-turbo-1106 | GPT-3.5 Turbo model with improved instruction following, JSON mode, reproducible outputs, parallel function calling, and more. Returns a maximum of 4,096 | 16,385 tokens | Up to Sep 2021 |

- *Sensibilidad a los datos:* Es importante gestionar cuidadosamente la privacidad y seguridad de los datos, ya que el modelo tiene acceso a la información ingresada por los usuarios durante las sesiones de chat.

La integración de GPT-3.5 Turbo en nuestro chatbot educativo ha sido esencial para ofrecer una experiencia de usuario avanzada y eficiente. Esta tecnología ha permitido implementar un sistema de respuesta intuitivo y altamente interactivo que puede manejar una variedad de consultas educativas. Sin embargo, es crucial manejar de manera efectiva los costos y la configuración del sistema para garantizar que el chatbot siga siendo económicamente viable y seguro para todos los usuarios.

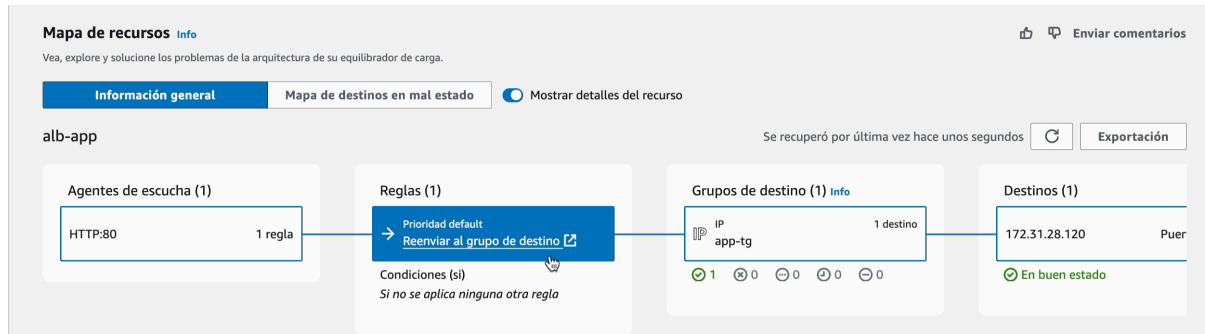
7.12.ANEXO: Configuración de Amazon ECS con Fargate y Application Load Balancer

En este proyecto, hemos utilizado Amazon ECS con la opción de Fargate para el despliegue del chatbot educativo, lo que nos permite operar sin tener que administrar servidores físicos. Este sistema utiliza una imagen de Docker alojada en Docker Hub (serhiyhs/chat-app:version3) para garantizar la consistencia y eficiencia del entorno de ejecución. Además, se integra un Application Load Balancer para manejar eficazmente el tráfico entrante.

The screenshot displays the AWS ECS console interface. At the top, a green banner indicates that the task definition was created correctly and the app service is implemented correctly. Below this, the main cluster summary shows ARN, state (Active), CloudWatch supervision, and registered container instances. Under the 'Servicios' tab, a table lists services by name, ARN, status, and deployment tasks. The 'app' service is selected, showing 1 task in execution and a deployment status of 'Completado'.

1. Configuración y Operación:

La imagen del chatbot se descarga automáticamente desde Docker Hub utilizando la definición de tarea en ECS, lo que facilita actualizaciones y mantenimiento continuo.



El ALB, configurado para escuchar en el puerto 80, redirige el tráfico al Target Group, que a su vez envía las solicitudes al puerto 8095 del contenedor ECS. Este puerto es específico para la aplicación del chatbot, lo que simplifica el enrutamiento y mejora la seguridad.



2. Grupos de Seguridad:

Para proteger nuestro sistema, hemos configurado grupos de seguridad específicos tanto para el ALB como para los servicios de ECS.

| Grupos de seguridad (3) <small>Información</small> | | | | | |
|---|--------------------------|-------------------------------|-----------------------|-------------------------------|--|
| <input type="button"/> Acciones <input type="button"/> Exportar los grupos de seguridad a CSV <input type="button"/> Crear grupo de seguridad | | | | | |
| <input type="text"/> Find resources by attribute or tag | | | | | |
| Name | ID de grupo de seguridad | Nombre del grupo de seguridad | ID de la VPC | Descripción | |
| - | sg-00ec977b29530cc8b | default | vpc-09dbf06e8e1aae73d | default VPC security group | |
| - | sg-00b50e4c0e33fb74f | alb-app-sg | vpc-09dbf06e8e1aae73d | Security Group del ALB de app | |
| - | sg-0e69e6539c581b107 | ecs-app-sg | vpc-09dbf06e8e1aae73d | Security Group de app ECS | |

| sg-0e69e6539c581b107 - ecs-app-sg | | | | | |
|--|--|---|--|-----------|----------------------|
| Detalles | | | | | |
| Nombre del grupo de seguridad <input type="checkbox"/> ecs-app-sg | ID del grupo de seguridad <input type="checkbox"/> sg-0e69e6539c581b107 | Descripción <input type="checkbox"/> Security Group de app ECS | ID de la VPC <input type="checkbox"/> vpc-09dbf06e8e1aae73d | | |
| Propietario <input type="checkbox"/> 843300104475 | Número de reglas de entrada <input type="checkbox"/> 1 Entrada de permiso | Número de reglas de salida <input type="checkbox"/> 1 Entrada de permiso | | | |
| Reglas de entrada | | | | | |
| <input type="text"/> Buscar | | | | | |
| Name | ID de la regla del gr... | Versión de IP | Tipo | Protocolo | Intervalo de puertos |
| - | sgr-0cf92a87cc90f3d55 | IPv4 | TCP personalizado | TCP | 8095 |

El grupo de seguridad de ECS restringe el tráfico de entrada solo a las solicitudes que provienen del ALB, permitiendo solo la comunicación a través del puerto 8095. Esto minimiza la superficie de ataque al limitar el acceso directo a los contenedores.

| Name | ID | Versión de IP | Tipo | Protocolo | Intervalo de puertos | Origen |
|------|-----------------------|---------------|------|-----------|----------------------|-----------|
| - | sgr-075df926e048a76cb | IPv4 | HTTP | TCP | 80 | 0.0.0.0/0 |

Por otro lado, el grupo de seguridad del ALB permite el tráfico entrante únicamente por el puerto 80 y bloquea cualquier tráfico no autorizado, asegurando que todas las respuestas sean procesadas de manera adecuada y segura.

3. Función del Target Group:

| Destinos totales | En buen estado | En mal estado | Sin utilizar | Inicial | Vaciado |
|------------------|----------------|---------------|--------------|---------|---------|
| 1 | 1 | 0 | 0 | 0 | 0 |
| | En buen estado | En mal estado | | | |
| | 0 Anómalo | | | | |

Distribución de destinos por zona de disponibilidad (AZ)
Seleccione los valores de esta tabla para ver los filtros correspondientes aplicados a la tabla Destinos registrados que aparece a continuación.

| Destinos | Monitorización | Comprobaciones de estado | Atributos | Etiquetas |
|----------|----------------|--------------------------|-----------|-----------|
|----------|----------------|--------------------------|-----------|-----------|

El Target Group es fundamental para administrar el tráfico dirigido por el ALB hacia los servicios de ECS. Su función es encaminar el tráfico entrante hacia las instancias apropiadas del contenedor basadas en la configuración de salud y rendimiento. En este caso, redirige todas las solicitudes entrantes al puerto 8095 de los contenedores donde opera la aplicación del chatbot. Este enfoque permite una gestión eficiente del tráfico y optimiza la disponibilidad y respuesta del chatbot al asegurar que las solicitudes sean manejadas solo por contenedores que están funcionando correctamente.

4. Ventajas y Beneficios de Seguridad:

Esta configuración no solo facilita la escalabilidad y el mantenimiento mínimo sino que también asegura una alta disponibilidad y una respuesta eficiente a las demandas del usuario. Los grupos de seguridad juegan un papel crucial en la protección de la infraestructura, filtrando el tráfico y previniendo accesos no deseados, lo que es esencial para mantener la integridad operativa y la seguridad de los datos del usuario.

La integración de ECS con Fargate y ALB, complementada con una estrategia de seguridad robusta, proporciona una solución potente y flexible para el despliegue de aplicaciones en la nube. Este enfoque no solo optimiza la gestión de recursos y la disponibilidad del servicio, sino que también mejora la protección contra amenazas externas, garantizando un entorno operativo seguro y eficiente para nuestro chatbot educativo.

7.13.ANEXO: Esquema Gantt actualizado

Inicialmente, el proyecto estaba planificado para seguir un desarrollo estructurado y ordenado, utilizando recursos avanzados de hardware y software y apoyándose en la infraestructura de AWS. Las tareas estaban distribuidas cuidadosamente en un diagrama de Gantt, con estimaciones de tiempo para cada fase, desde el diseño hasta la evaluación final. Se esperaba un flujo de trabajo continuo y sin mayores obstáculos, basado en una utilización constante de las herramientas de desarrollo y servicios cloud elegidos.

Durante la implementación del proyecto, surgieron varios desvíos significativos respecto a lo planificado:

- Limitaciones de Recursos: Los recursos de laboratorio gratuito presentaron más limitaciones de lo anticipado, afectando tanto la disponibilidad de servicios de AWS como la capacidad de ejecución continua del chatbot.
- Problemas Técnicos Inesperados: Aparecieron problemas técnicos, especialmente con la gestión de sesiones simultáneas y la estabilidad del entorno de desarrollo, lo que requirió la implementación de soluciones improvisadas que no estaban en el plan original.
- Delays en la Cronología: Las etapas de prueba y ajuste necesitaron más tiempo del estimado debido a los imprevistos técnicos y la necesidad de iteración basada en pruebas continuas.
- Adaptaciones en el Desarrollo: Las funcionalidades del chatbot tuvieron que ser adaptadas para cumplir con los requerimientos reales de desempeño bajo las restricciones del entorno de prueba, lo cual no estaba completamente previsto en la fase de diseño inicial.

A pesar de seguir una planificación detallada y contar con tecnologías avanzadas, el proyecto tuvo que adaptarse a las realidades del entorno de desarrollo y los imprevistos. Esto llevó a modificaciones significativas en las fases de desarrollo, pruebas y despliegue, destacando la importancia de la flexibilidad y la capacidad de respuesta rápida a los obstáculos. Las revisiones en la gestión de proyecto fueron cruciales para alinear las expectativas con los resultados prácticos, demostrando que en proyectos tecnológicos dinámicos, adaptarse a los cambios es tan importante como la planificación inicial.

| | | | | | | |
|----|---|----------|----------|----------|---------------------------|-----|
| 1 | Anteproyecto 21/03/2024 | 15/03/24 | 21/03/24 | Complete | Aceptado | 7d |
| 2 | Búsqueda de Ideas para el anteproyecto | 15/03/24 | 16/03/24 | Complete | | 2d |
| 3 | Realizar investigación para el anteproyecto | 18/03/24 | 19/03/24 | Complete | | 2d |
| 4 | Realizar el anteproyecto | 20/03/24 | 21/03/24 | Complete | | 2d |
| 5 | Realizar la entrega del anteproyecto | 21/03/24 | 21/03/24 | Complete | | 1d |
| 6 | | | | | | |
| 7 | Proyecto (Entrega 1) 14/04/2024 | 22/03/24 | 14/04/24 | Complete | Calificación 7,25 / 10,00 | 24d |
| 8 | Planificación adecuado de tareas | 22/03/24 | 24/03/24 | Complete | | 3d |
| 9 | Realizar investigación para el proyecto | 27/03/24 | 29/03/24 | Complete | | 3d |
| 10 | + Empezar a documentar el proyecto (este apartado se irá ampliando en cada entrega) | 30/03/24 | 09/04/24 | Complete | | 11d |
| 17 | Empezar con el diseño de la pagina WEB y ChatBot | 02/04/24 | 11/04/24 | Complete | | 10d |
| 18 | Realizar primera entrega del proyecto | 14/04/24 | 14/04/24 | Complete | | 1d |
| 19 | | | | | | |
| 20 | Proyecto (Entrega 2) 05/05/2024 | 19/04/24 | 06/05/24 | Complete | Calificación 9,00 / 10,00 | 18d |
| 21 | + Realizar correcciones de la entrega 1 | 19/04/24 | 22/04/24 | Complete | | 4d |
| 26 | Completar el Glosario (este apartado se irá ampliando en cada entrega) | 23/04/24 | 24/04/24 | Complete | | 2d |
| 27 | Esquema de interconexión | 25/04/24 | 26/04/24 | Complete | | 2d |
| 28 | Especificaciones de la infraestructura | 27/04/24 | 28/04/24 | Complete | | 2d |
| 29 | Descripción funcionalidades | 22/04/24 | 01/05/24 | Complete | | 10d |
| 30 | Diseño (diseño de la aplicación en caso de tener interfaz gráfica) | 20/04/24 | 29/04/24 | Complete | | 10d |
| 31 | Esquema BBDD - Diagrama E/R | 04/05/24 | 04/05/24 | Complete | | 1d |
| 32 | Realizar la entrega del proyecto | 05/05/24 | 06/05/24 | Complete | | 2d |
| 33 | | | | | | |
| 34 | Proyecto (Entrega 3) 19/05/2024 | 05/05/24 | 19/05/24 | Complete | Calificación 8,13 / 10,00 | 15d |
| 35 | Realizar correcciones de la entrega 2 | 05/05/24 | 07/05/24 | Complete | | 3d |
| 36 | Inicio Plan de Pruebas | 08/05/24 | 10/05/24 | Complete | | 3d |
| 37 | Inicio conclusiones | 11/05/24 | 13/05/24 | Complete | | 3d |
| 38 | Inicio documentar anexos | 14/05/24 | 16/05/24 | Complete | | 3d |
| 39 | Código / Configuración | 17/05/24 | 19/05/24 | Complete | | 3d |
| 40 | Realizar la entrega del proyecto | 19/05/24 | 19/05/24 | Complete | | 1d |
| 41 | | | | | | |
| 42 | Proyecto (Entrega 4) 02/06/2024 | 19/05/24 | 02/06/24 | Complete | | 15d |
| 43 | + Realizar correcciones de la entrega 3 | 20/05/24 | 25/05/24 | Complete | | 5d |
| 47 | Ultimos ajustes | 26/05/24 | 29/05/24 | Complete | | 3d |
| 48 | Ultimas pruebas | 30/05/24 | 01/06/24 | Complete | | 3d |
| 49 | Corrección de errores detectados | 30/05/24 | 01/06/24 | Complete | | 3d |
| 50 | Realizar la entrega del proyecto | 02/06/24 | 02/06/24 | Complete | | 1d |

Esquema 15: Diagrama de Gantt-SmartSheet actualizada.

Memoria de Trabajo Final de Grado

Implementación de un ChatBot

| Tasks | Mar 10 | Mar 17 | Mar 24 | Mar 31 | Apr 7 | Apr 14 | Apr 21 | Apr 28 | May 5 | May 12 | May 19 | May 26 | Jun 2 | |
|---|--------|--------|--------|--------|-------|--------|--------|--------|-------|--------|--------|--------|-------|---|
| | D | L | M | M | J | V | S | D | L | M | M | J | V | S |
| Anteproyecto 21/03/2024 | | | | | | | | | | | | | | |
| Busqueda de ideas para el anteproyecto | | | | | | | | | | | | | | |
| Realizar investigación para el anteproyecto | | | | | | | | | | | | | | |
| Realizar el anteproyecto | | | | | | | | | | | | | | |
| Realizar la entrega del anteproyecto | | | | | | | | | | | | | | |
| Proyecto (Entrega 1) 14/04/2024 | | | | | | | | | | | | | | |
| Planeación adecuada de tareas | | | | | | | | | | | | | | |
| Realizar investigación para el proyecto | | | | | | | | | | | | | | |
| + Empezar a documentar el proyecto (este apartado se irá ampliando en cada entrega) | | | | | | | | | | | | | | |
| + Empiezar con el diseño de la página WEB y ChatBot | | | | | | | | | | | | | | |
| Realizar primera entrega del proyecto | | | | | | | | | | | | | | |
| Proyecto (Entrega 2) 05/05/2024 | | | | | | | | | | | | | | |
| + Realizar correcciones de la entrega 1 | | | | | | | | | | | | | | |
| Complejar el Glosario (este apartado se irá ampliando en cada entrega) | | | | | | | | | | | | | | |
| Esquema de interconexión | | | | | | | | | | | | | | |
| Especificaciones de la infraestructura | | | | | | | | | | | | | | |
| Descripción funcionalidades | | | | | | | | | | | | | | |
| Diseño (diseño de la aplicación en caso de tener interfaz gráfica) | | | | | | | | | | | | | | |
| Esquema BBDD - Diagrama ER | | | | | | | | | | | | | | |
| Realizar la entrega del proyecto | | | | | | | | | | | | | | |
| Proyecto (Entrega 3) 19/05/2024 | | | | | | | | | | | | | | |
| Realizar correcciones de la entrega 2 | | | | | | | | | | | | | | |
| Inicio Plan de Pruebas | | | | | | | | | | | | | | |
| Inicio conclusiones | | | | | | | | | | | | | | |
| Inicio documentar annexos | | | | | | | | | | | | | | |
| Código / Configuración | | | | | | | | | | | | | | |
| Realizar la entrega del proyecto | | | | | | | | | | | | | | |
| Proyecto (Entrega 4) 02/06/2024 | | | | | | | | | | | | | | |
| + Realizar correcciones de la entrega 3 | | | | | | | | | | | | | | |
| Últimos ajustes | | | | | | | | | | | | | | |
| Últimas pruebas | | | | | | | | | | | | | | |
| Corrección de errores detectados | | | | | | | | | | | | | | |
| Realizar la entrega del proyecto | | | | | | | | | | | | | | |